POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# Requirement Analysis and Specification

Computer Science and Engineering

Author(s): **Giacomo Orsenigo**
**Federico Saccani**
**Francesco Ferlin**

Academic Year: 2023-24

| | |
|---|---|
| **Deliverable:** | RASD |
| **Title:** | Requirement Analysis and Specification Document |
| **Authors:** | Giacomo Orsenigo, Federico Saccani, Francesco Ferlin |
| **Version:** | 1.0 |
| **Date:** | 21-12-2023 |
| **Download page:** | https://github.com/Furrrlo/FerlinOrsenigoSaccani/ |
| **Copyright:** |  |

# Contents

# 1 Introduction

## 1.1. Purpose

The purpose of the system is to help students improve their software development skills by joining Code Kata Battles. A Code Kata Battle is a programming exercise created by an educator that contains a set of test cases. Students have to provide a solution that passes the tests, using a test-driven approach. Students can try as many times as they want, each time a solution is submitted, the tests are executed and the score is calculated.

The goals of the CKB platform are:

**G1**: Educators can organize tournaments of programming exercises aimed at improving their students' skills.

**G2**: Educators can organize battles in a tournament they have created.

**G3**: Educators can aid their colleagues in creating battles.

**G4**: Educator can decide how to score a battle they have created.

**G5**: Students can be notified about new tournaments.

**G6**: Students can subscribe to tournaments.

**G7**: Students can be notified about new battles in tournaments they are subscribed to.

**G8**: Students can form teams to take part in a battle.

**G9**: Teams of students can join a battle.

**G10**: Students and educators can monitor the progress of all students in battles and tournaments.

**G11**: Educators can manage students' gamification badges.

**G12**: Students can receive badges depending on the rules they have fulfilled.

**G13**: Students and educators can see badges collected by a student.

## 1.2. Scope

The platform is intended to be used in a school or university context. Educators are the professors who teach in the school.
The platform interacts with an existing login system (SSO), responsible for authenticating users.

The platform will allow an educator to create tournaments and battles within the context of a tournament he has created. The educator can also grant other colleagues the permission to create battles within the context of his tournaments.

The creator of the tournament can also create badges. Each badge has a set of rules that must be fulfilled to achieve it. The rules are simple expressions on variables that returns a boolean value. For example:

```
tot_attended_battles > 0
```

or

```
tot_commits_student == max_tot_commits
```

where tot_attended_battles, tot_commits_student and max_tot_commits are pre-defined variables.
To create more complex rule and avoid duplicated code, educators can create new variables. In particular, they can use an interpreted language to define new variables based on the existing ones. For example, if the platform provides the variables tot_attended_battles and tot_battles, educators can create a new variable tot_missed_battles as

```
var tot_missed_battles = tot_battles - tot_attended_battles;
```

To create a battle, the educator needs to upload test cases and build automation scripts. Uploading test cases he can decide if each test is public or private. Public tests are provided to students, so that they can run them locally. Private tests are used for automatic evaluation, but they are not provided to students. In this way educator are sure that students are not writing code that only pass specific tests, but actually solves the exercise.

When the registration deadline of a battle expires, the platform creates a GitHub repository containing build automation scripts and public tests provided by the educator. All subscribed students receive a notification. One student per group forks the repository and adds his teammates as collaborators.
The forked repo contains also a workflow that triggers the CKB platform on each push. The student needs to enable the GitHub Actions to make it work. The student also has to insert the link of his repository in the CKB platform, so that only authorized repositories can trigger the platform.

When the submission deadline of a battle expires, if the educator enabled manual evaluation, there is a consolidation stage. During this stage, the educator who create the battle uses the platform to go through the sources produced by each team to assign his/her score. After grading all groups, he can close the consolidation stage.
Otherwise, if manual evaluation is not required, when the submission deadline expires, the battle is automatically closed.

After the battle is closed, the final battle rank becomes available and all students who participated are notified.

After all battles in a tournament are closed, the educator who create the tournament can close it.

The following table describes world, shared and machine phenomena.

| Phenomena | Controlled by | Shared |
|---|---|---|
| A user wants to log in to the platform | W | N |
| An educator wants to create a tournament | W | N |
| A student wants to take part in a tournament | W | N |
| A student wants to invite other students in a tournament | W | N |
| A student wants to see the ranking in a battle | W | N |
| An educator wants to see the ranking in a battle | W | N |
| A user logs in | W | Y |
| An educator creates a tournament | W | Y |
| The system notifies students about a new tournament | M | Y |
| An educator grants permissions for a tournament to another educator | W | Y |
| A student subscribes to a tournament | W | Y |
| An educator creates a battle as part of a tournament he has permissions for | W | Y |
| The system notifies students subscribed to a tournament about a new battle in said tournament | M | Y |
| A student invites another student to take part in a battle with him | W | Y |
| A student accepts the invitation of another student to take part in a battle with him | W | Y |
| A student joins a battle on their own | W | Y |
| The registration deadline of a battle expires | W | N |

| | | |
|---|---|---|
| The system sends the link to a GitHub repository containing the code for a battle to all students who are members of subscribed teams | M | Y |
| A student forks the GitHub repository | W | N |
| A student sets up the automated workflow | W | N |
| A student writes code | W | N |
| A student pushes a new commit in their repository main branch | W | Y |
| The system pulls the latest sources, analyses them, and runs the tests | M | N |
| The system calculates the battle score of the team. | M | N |
| A student or an educator visualize the current rank of a battle | W | Y |
| The submission deadline expires and the consolidation stage starts | W | N |
| An educator manually evaluates the submission of a team for a battle he has created | W | Y |
| An educator closes the consolidation stage | W | Y |
| The system notifies students taking part in a battle that the final battle rank is available | M | Y |
| An educator closes a tournament | W | Y |
| The platform updates the personal tournament score of each student | M | N |
| The system notifies students taking part in a tournament that the final tournament rank is available | M | Y |
| Any subscribed user visualizes the list of current tournaments | W | Y |
| Any subscribed user visualizes the ranking of a tournament | W | Y |
| An educator defines a badge | W | Y |
| A student achieves a badge | W | Y |
| Any subscribed user visualizes the badges achieved by a student | W | Y |

Table 1.1

## 1.3. Definitions, Acronyms, Abbreviations

### 1.3.1. Definitions

**CodeKataBattle (CKB):** A platform designed to help students enhance their software development skills through collaborative coding exercises known as code kata battles.

**Kata:** In karate, a kata is an exercise that involves repetitive practice to improve coding skills. Code Kata brings this concept to software development as a form of deliberate practice.

**Test-First Approach:** A development methodology in which developers write tests before implementing the corresponding code, ensuring that the code meets specified requirements and functions as expected.

**GitHub:** A web-based platform that provides version control and collaboration features using Git, facilitating software development projects. [3]

**API (Application Programming Interface):** A set of rules and protocols that allows different software applications to communicate and interact with each other.

**Static Analysis Tools:** Software tools that analyze source code without execution, providing insights into code quality, security, reliability, and maintainability.

**Gamification Badges:** In the context of the CKB platform, these are rewards representing individual achievements within tournaments. Educators define badges with titles and rules.

**Tournament:** A series of Code Kata Battles organized by educators on the CKB platform, providing students with opportunities to compete and improve their coding skills.

**Battle:** A coding exercise within a tournament that includes a textual description, a software project with test cases, and build automation scripts. Teams aim to complete the project by following a test-first approach.

**Deadline:** A specified date or time by which certain actions, such as registration or submission, must be completed.

**Score:** A numerical representation of a team's performance in a battle, determined by various factors such as test case success, timeliness, and code quality.

**Rank:** The position of a team or student relative to others in a competition, often determined by their scores or performance.

**Notification:** A message or alert that informs users about events, changes, or updates within the platform.

**Collaborators:** Educators who are granted permission to create battles within a specific tournament.

**Build Automation Scripts:** Scripts that automate the process of building, compiling, and testing code.

**Git:** A distributed version control system used for tracking changes in source code during software development. [2]

**Git Repository:** A storage location for a Git project, containing all files and the version history of those files.

**GitHub Actions:** Automated workflows defined on GitHub that can perform various tasks such as building, testing, and deploying projects.

**Personal Tournament Score:** The cumulative score of a student in all battles within a specific tournament, reflecting their overall performance.

**Gamification:** The application of game elements, such as badges and rewards, in non-game contexts to motivate and engage users.

**API Calls:** Requests made by the platform to external services, such as GitHub, using predefined interfaces to retrieve or send data.

**Dynamic Ranking:** A continuously updated ranking that changes based on ongoing events or activities, providing real-time insights into participants' standings.

### 1.3.2. Acronyms

**CKB:** CodeKataBattle

**API:** Application Programming Interface

**SAT:** Static Analysis Tools

**GAB:** Gamification Badges

**BAS:** Build Automation Scripts

**GR:** Git Repository

**GH:** GitHub

**PTS:** Personal Tournament Score

**DR:** Dynamic Ranking

**DL:** Deadline

**TDD:** Test-Driven Development

**SSO:** Single Sign On

**IDE:** Integrated Development Environment

### 1.3.3. Abbreviations

**e.g.:** For example

**e.g.:** That is

**repo:** Repository

**Gn:** Goal number n

**Dn:** Domain assumption number n

**Rn:** Requirement number n

**UCn:** Use case number n

## 1.4. Revision history

- **1.0** - Initial release (21st December 2023)

## 1.5. Reference Documents

**Specification document:** *"Assignment RDD AY 2023-2024"*

**ISO/IEC/IEEE 29148 (Nov 2018)** - International Standard - Systems and software engineering – Life cycle processes – Requirements engineering:
`https://doi.org/10.1109/IEEESTD.2018.8559686`

**The world and the machine** - Michael Jackson, 1995:
`https://doi.org/10.1145/225014.225041`

**UML official specification:** `https://www.omg.org/spec/UML/`

**Alloy official documentation:** `https://alloytools.org/documentation.html`

## 1.6. Document Structure

1. **Section 1: Introduction**
   This section gives a brief description of the problem, as well as the purpose and the scope of the system. It also contains the list of definitions, acronyms and abbreviations that might be encountered while reading the document. Additionally, there's the revision history of the document, which keeps track of the various version, their release date and their changes.
2. **Section 2: Overall Description**
   This section contains a description of the entirety of the problem, detailing its domain, including assumptions that must hold true, as well as how its state changes in time and possible usage scenarios.

3. **Section 3: Specific Requirements**
   This section describes the specific requirements of the system. It provides a detailed analysis of the external interfaces, both user-facing as well as hardware and software, and presents the performance requirements, standards compliance and software system attributes. Additionally, it describes in depth the functional requirements of the system and details the use cases by means of diagrams, mapping goals, requirements and domains and tracing each use case to the requirements it satisfies.

4. **Section 4: Formal Analysis using Alloy**
   This section provides a formal analysis using the alloy language, in order to prove the correctness and soundness of the system described in the previous sections.

# 2 | Overall Description

## 2.1. Product perspective

### 2.1.1. Scenarios

1. **Educator creates a new Tournament**
   Luca Proserpio is an educator who works for a famous Italian university and is subscribed to the CKB Platform.
   Luca wants to create a new tournament on the platform so that students can participate in it to improve their programming skills in Java.
   Luca logs in, and the homepage will be shown.
   Luca among the various options on the homepage decides to click on the *"Create New Tournament"* button.
   The platform shows a form which contains:

   (a) Name field: to set the name of the tournament

   (b) Collaborators list: with all the educators in the platform

   (c) Deadline field: to define the deadline date to be able to register for the tournament

   (d) Create badges button: to create new badges

   (e) Create Tournament button: to proceed with the creation of the tournament

   Luca enters the name *"Welcome Tournament School Year 2024"* in the Name field and sets the date 30/09/2023 in the Deadline field.
   Next, he clicks on the list of educators selecting educators Gianpaolo Mariani, Mario Rossi, and Rosa Ballabio as he wants to allow them also to be able to create new battles for the tournament he is creating.
   Accordingly, he clicks on the Create Tournament button and the platform shows a message of successful tournament creation. Luca will be shown the details of the newly created *"Welcome Tournament School Year 2024"* and the list of battles available in this tournament (empty at the moment). All students enrolled in the CKB Platform will be notified about the new creation of the tournament created by educator Luca.

2. **Educator creates new badges**
   Luca Proserpio is an educator and he is creating the *"Welcome Tournament School Year 2024"* tournament. Luca wants to create three badges, to encourage students to participate.

He clicks the *"Create badges"* button to create the first badge. He enters the title *"Start2024_with_the_right_foot"* and he does not enter any particular rule because he wants the badge to be assigned to all enrolled students.
Luca clicks the *"add"* button to create the second badge. He enters the title *"Participant_2024"* and he writes a rule specifying that the predefined variable *tot_commits_student* >= 1.
Luca clicks the *"add"* button another time and enter the title *"Best_Participants_2024"* to award students who won at least 2 battles. The platform provides a predefined array variable *"final_positions_student"* that represent the final position of the student in each battle. For simplicity, Luca decide to create a new variable *battlesWon* defined as

```
var battlesWon = 0;
for (let pos of final_positions_student) {
        if (pos == 1)
                battlesWon++;
}
```

Then he creates a rule specifying the condition *battlesWon >= 2*. Luca clicks the confirmation button to create the badges.

3. **Educator creates a new Battle for an Existing Tournament**
   Gianpaolo Mariani is an educator registered with the CKB Platform and wants to create a battle for the *"Welcome Tournament School Year 2024"* tournament.
   After logging in, the platform shows the list of tournaments he has created or to which he has been added by other contributors. Gianpaolo clicks on the *"Welcome Tournament School Year 2024"* line found in the list of tournaments mentioned earlier.
   After clicking, the platform shows the details of the selected tournament and a button to create a new battle. Gianpaolo clicks on the *"Create New Battle"* button and the platform shows a form containing:

   (a) Kata Code Upload Section

      i. Battle Description field

      ii. Form to upload test cases

      iii. Form to upload build automation scripts

   (b) Field group policy: to set minimum and maximum number of students per group

   (c) Field registration deadline: to define maximum date for which students can register for the battle

   (d) Field final submission deadline: to define maximum date for which students can submit code to be evaluated

(e) Scoring Configuration section

i. Form functional aspects

ii. Form timeliness

iii. Form quality level

iv. Form optional manual evaluation

(f) Create Battle button

Gianpaolo fills in all the fields by entering the description of the battle, the registration deadline as 6/11/2023, the final submission deadline as 23/12/2023, the minimum and maximum number of students as 3, and does not edit any of the default entries in the Scoring Configuration section as he plans to edit it later before the battle is actually started.
After that, he clicks on the Create Battle button and the platform redirect Gianpaolo to the tournament details, that contain the new created battle in the list of available battles. All students who had signed up for the tournament will receive a notification of the newly added battle created by educator Gianpaolo.

4. **Student joins to an existing Tournament by receiving a notification**
Marco is a university student and thus he is registered on the CKB Platform. He wants to sign up for a tournament to improve his programming skills in Java as he notices that he has difficulty writing code using Object-Oriented Programming.
In the afternoon Marco receives a notification of a new tournament creation called *"The Basics of Object-Oriented Programming in Java"*, he decides to take the opportunity and sign up.
Therefore, he accesses the platform by logging in and the platform redirect him to the list of available tournaments. Marco clicks on the tournament *"The Basics of Object-Oriented Programming in Java"*, for which he is not subscribed.
The platform shows Marco the information about the name and the description of the selected tournament.
Marco clicks on the Subscribe button, and the platform enrolls Marco in the tournament showing a confirmation registration message.

5. **Students create a team for a tournament battle**
Marco, Stefano, and Carlo are students enrolled in *"Welcome Tournament School Year 2024"* tournament.
They want to participate in one of the available tournament battles by creating a team. Specifically, Marco logs in and clicks on the tournament from the list of tournaments in which he is registered.
Marco is shown the details of the tournaments and the list of available battles within it.
He selects the first available battle and clicks on the *"Participate by creating a team"*

button.
Marco already knows Stefano and decides to invite him to join his team, Stefano receives the notification of the invitation and clicks it to accept right away.
Because the tournament has a minimum requirement of 3 people, Marco leaves the page, in hopes to find another student who wants to participate.
At a later time, Marco comes back on the page and invites Carlo, which he has just met during a lecture.
Carlo misses the notification, so he logs in and finds the invitation. He clicks on the *accept* button.
Once Carlo has accepted, Marco goes back to the previous page, where he can see the team he has formed, and clicks on the *"Participate with the current team"* button.

6. **Student forks the repository**
   Marco is a student and is participating in a battle in *"Welcome Tournament School Year 2024"* with his team. The system, when the registration deadline of the battle expires, creates the GitHub repository containing the code kata specified by the educator and sends the link of the newly created repository to all students who have registered. Marco receives the notification and decides, in agreement with his team, to fork the repo. Then he adds his teammates as collaborators in the GitHub repository.
   To make automatic evaluation work, he enables GitHub Action. Then he logs in to the CKB platform and inserts the link of his group's repository.

7. **Students receive a notification about the new team battle and personal students score when a new push to the forked GitHub repository is performed**
   Marco is a student and is participating in a battle with the team consisting of Stefano and Carlo.
   Marco edits the project code and commits and pushes to the forked GitHub repo.
   The platform fetches the code, analyzes it, and runs the prepared tests (test cases) for the battle in which the team is participating.
   The platform updates the team's battle score by considering:

   (a) The number of successfully passed test cases

   (b) The number of days that have passed since the end of the registration deadline

   (c) The quality level of the sources extracted from the static analysis tools

   Therefore, after the update, Marco, Stefano and Carlo receive a notification of the update.
   Marco logs in, go to the details of the battle and view the new battle score assigned to him automatically by the platform.

8. **Educator manually evaluate teams when the Consolidation Stage begins**
   Gianpaolo Mariano is a platform educator and had created a battle for the *"Welcome Tournament School Year 2024"* with final submission deadline as 23/12/2023.
   The platform at the end of the final submission deadline declares the start of the consolidation stage due to the fact that Gianpaolo had selected the *"manual evaluation"* option for the battle he created.
   Gianpaolo proceeds by hand analyzing the code submitted by the various groups and, in particular, decides to increase the score by +2 obtained by the group of students composed of Marco, Stefano and Carlo.
   Afterwards, Gianpaolo using the platform declares the consolidation stage closed.
   Marco, Stefano and Carlo and all other students who participated in the tournament battle receive a notification with the final battle rank.
   The platform also modifies each student's personal tournament score by summing all the battle scores received in that tournament.

9. **Educator closes a tournament**
   Luca Proserpio is a platform educator who had created the *"Welcome Tournament School Year 2024"* tournament.
   In March 2024, Luca decides to close the tournament.
   All students enrolled in the CKB Platform will be notified about the closure of the Tournament.
   Gianpaolo had also created the following gamification badges:

   (a) *"Start2024_with_the_right_foot"*: awarded to all students enrolled in the tournament.

   (b) *"Participant_2024"*: awarded to all students who have made at least 1 commit in any battle present in the tournament

   The platform analyzes all the details of the students who had registered for the tournament and accordingly assigns all students who met the requirements the relevant badges created by Luca.
   In particular, Marco, Stefano, and Carlo had participated in several battles in the tournament and therefore receive both badges.
   Samuele is another student who signed up for the tournament but never participated in any battle within the tournament, as a result the platform only assigns him the *"Start2024_with_the_right_foot"* badge.

### 2.1.2. Domain class diagram

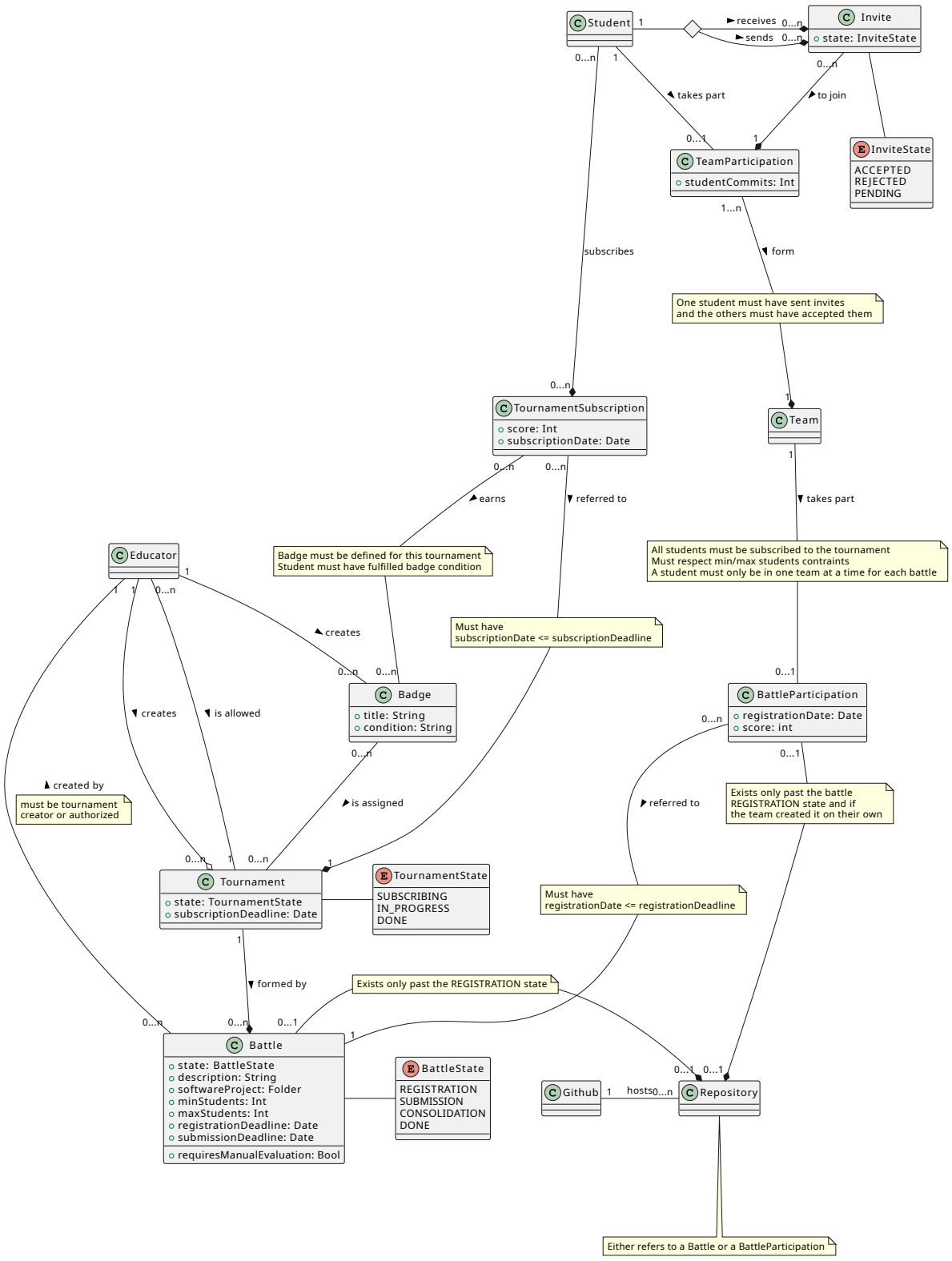



Figure 2.1: Domain class diagram

### 2.1.3. Statecharts

The following discusses how the state of a tournament and of a battle evolves in time, in order to have a better understanding of their lifecycle in the model.

**SD1**: **Tournament**

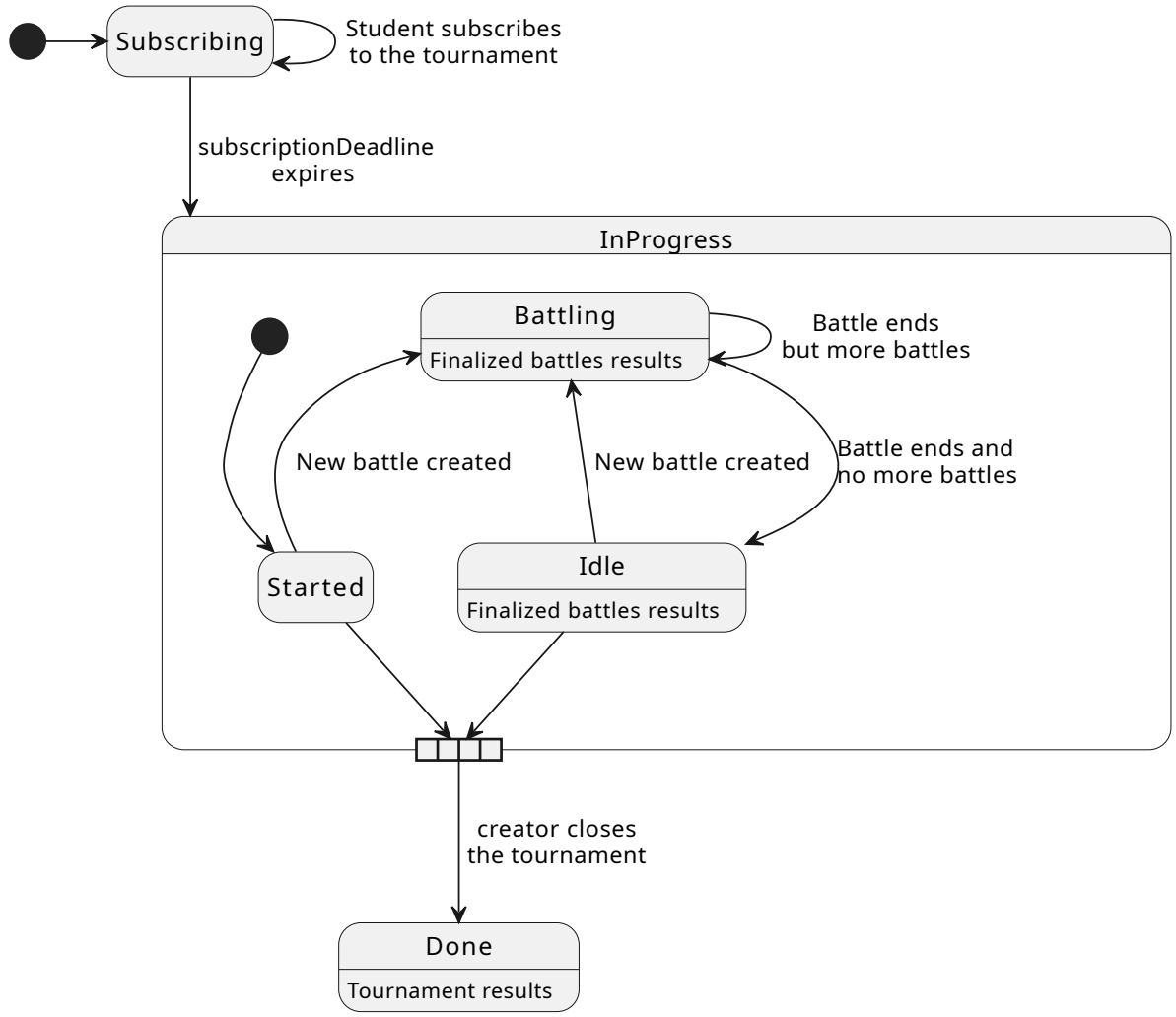



Figure 2.2: Tournament state diagram

When a tournament is created by an educator, it starts in a Subscribing state. During this phase, students can subscribe to the tournament so that they will be able to participate to its battles once it starts.
When the subscription deadline which was set by the creator at the creation of the tournament expires, it enters the InProgress state. During this phase, authorized educators can create new battles and students which previously subscribed are able to take part into them.
Once there are no more battles in progress (in the picture the Idle substate of the InProgress state), the educator which created the tournament can decide to close the tournament: once that is done, the tournament results and score is calculated and finalized.

**SD2: Battle**

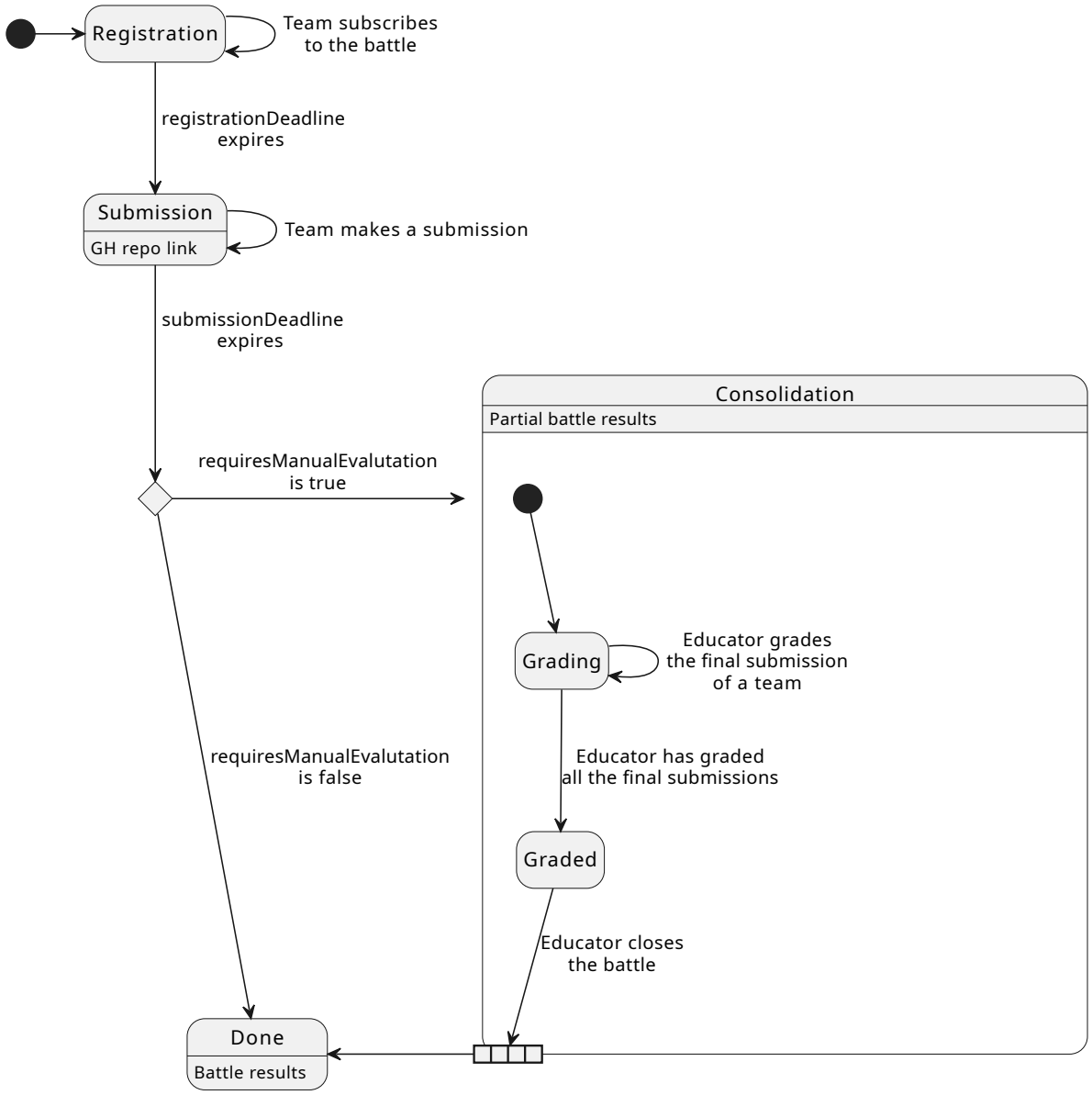



Figure 2.3: Battle state diagram

When a battle is created by an authorized educator, it starts in the Registration state. During this phase, teams of students can register to the battle so that they will be able to make submissions once it starts.
When the registration deadline which was set by the educator at the creation of the battle expires, it enters the Submission state. During this phase, students can fork and clone the GH repo of the battle and make submissions.
When the submission deadline, which was also set by the educator at the creation time, expires the battle enters either the Consolidation or the Done state, depending on whether manual evaluation is required.
In the Consolidation state, the educator manually grades the final submission of each team. Once that is done, it can manually close the battle, so that it goes in the Done state.

## 2.2. Product functions

### 2.2.1. Automatic evaluation

The main function of the system is to allow student solving programming exercises, evaluating their solutions. After joining a battle, students can clone the forked repository and start programming using the favorite text editor or IDE. The forked repo contains build automation scripts and public test cases uploaded by the educator, so that students can try running their code. When they want to be evaluated, they can commit and push their work. In this way, the GitHub action will start and trigger the CKB platform. The platform will clone the repository and build the program using the build automation script. If the build succeeds, the system starts the automated evaluation tasks:

- Run public and private tests
- Compute the time elapsed since the start of the battle
- Analyze code with a static analysis tool

Based on the results of these steps, the system assigns a grade, that students can see in their personal area. The GitHub repository also allows students to collaborate with their teammates, possibly using different branches to implement the required features.

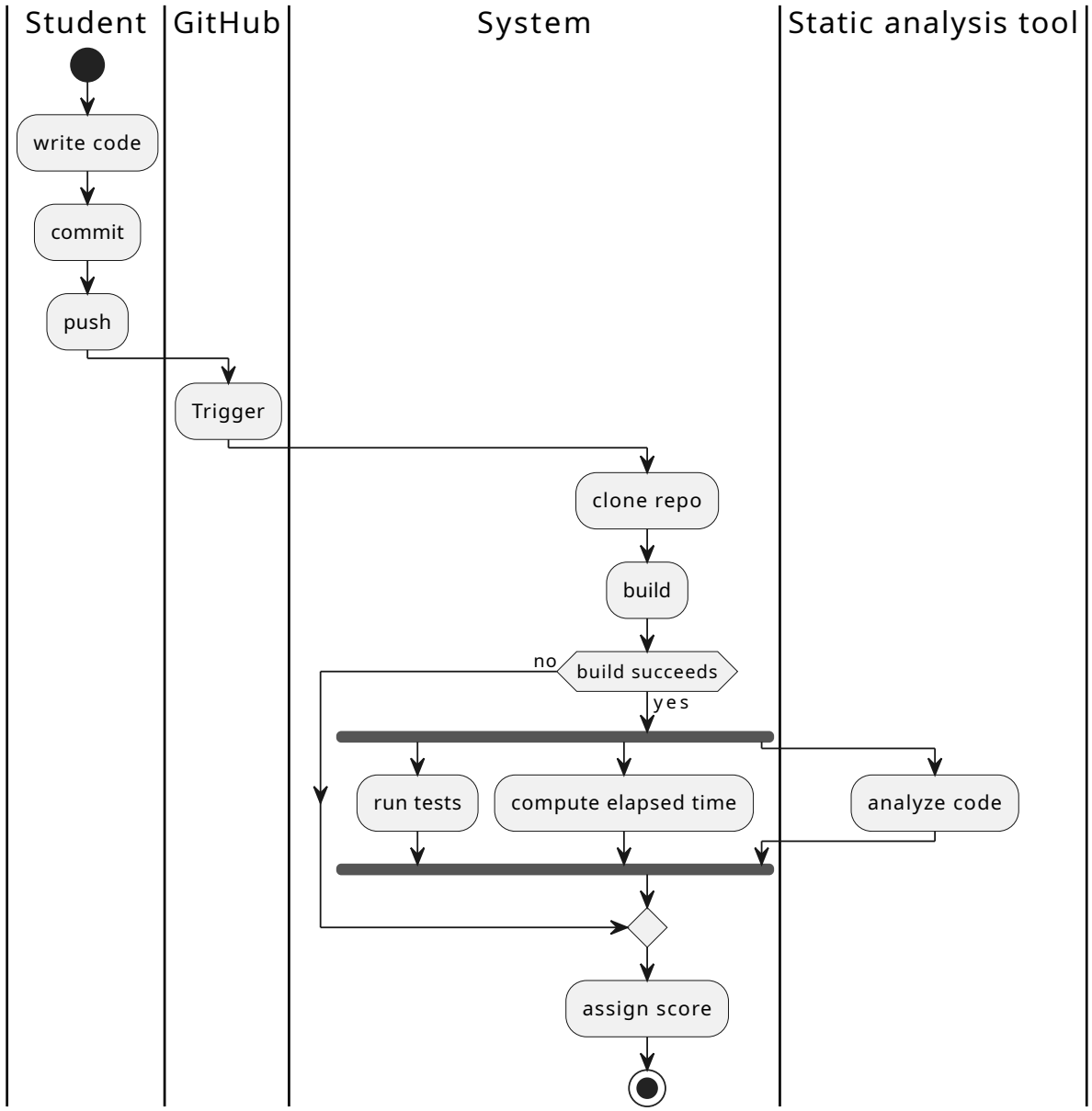



Figure 2.4: Automatic evaluation activity diagram

### 2.2.2. Battle evaluation

During the consolidation stage, the educator can access the GitHub repository of each team. The git repo also allows the educator to see who write each line of code, through the git blame tool. This allows the educator to assign different grades to students of the same team, based on the work done by each student. The educator uses the CKB platform to manually insert the grades. After grading all groups, he can close the consolidation stage. Otherwise, if manual evaluation is not required, when the submission deadline expires, the battle is automatically closed. After the battle is closed, the platform calculates the final ranks and notifies participating students. The platform also updates the personal tournament score of each student, adding the score of the current battle.

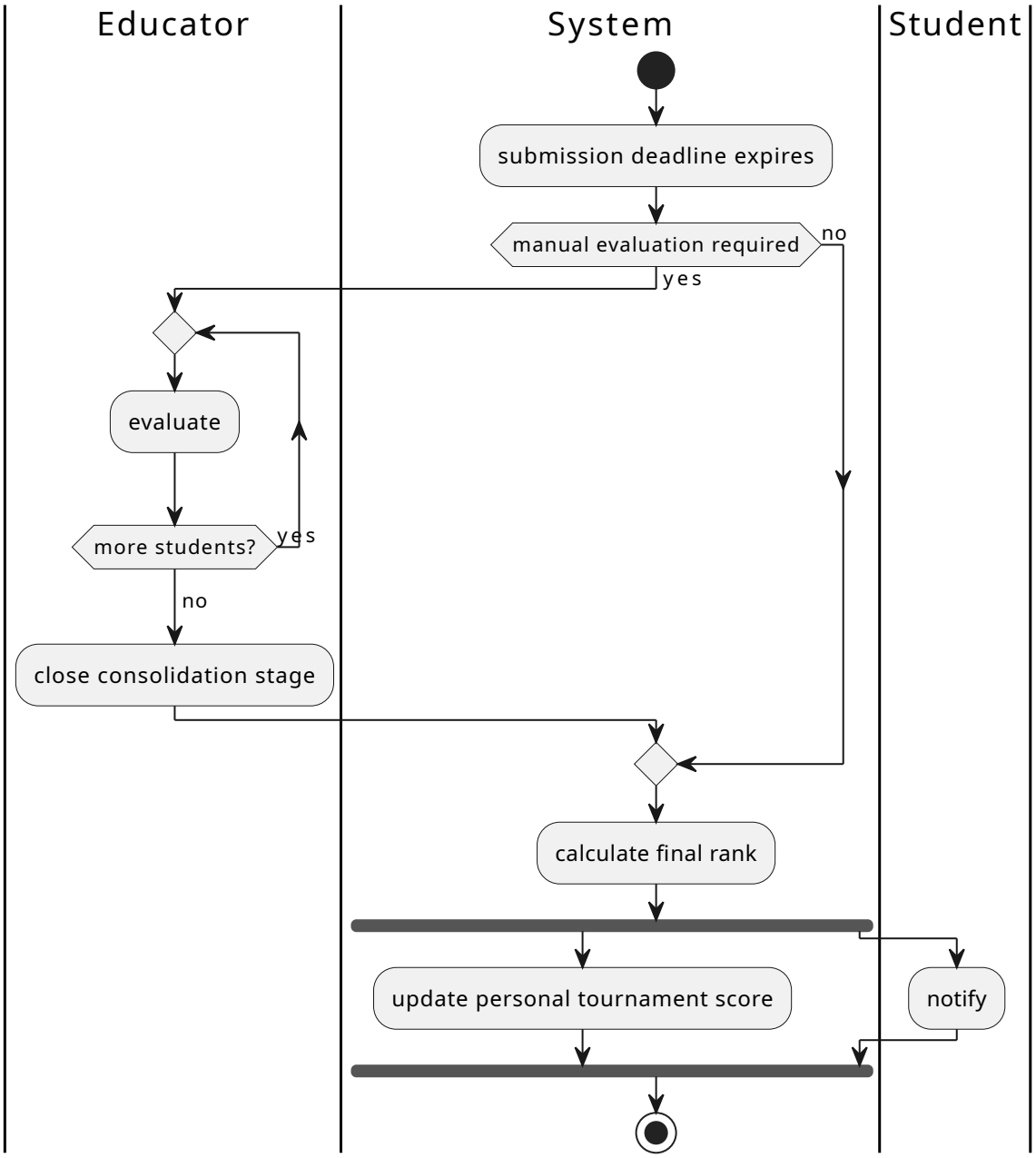



Figure 2.5: Battle evaluation activity diagram

## 2.3. User characteristics

### 2.3.1. Student

The student is a client, i.e. a person who is able to access the CKB Platform.
They use the platform with the goal of honing their software development skills. Their participation takes place through several stages: they begin by signing up for tournaments and code kata battles, where they form or join teams of developers. Once involved, teams tackle a series of programming exercises in different languages.
Each student participates in tournaments within which he or she conducts battles and obtains scores.
They receive updates on battle scores, view final standings and personal tournament scores. Can also view and collect badges obtained during competitions.

### 2.3.2. Educator

The educator is a teacher who is able to access the CKB Platform. They use the platform with the goal of running tournaments and code kata battles, actively contributing to the students' educational process.
Its involvement follows several stages:
It begins by creating tournaments, defining the specifics of the battles, and establishing clear rules for evaluating students' proposed solutions with the ability to manually evaluate students' work.
An educator also manages gamification badges, i.e. rewards that provide additional incentive for students to succeed in their software development efforts.
His or her participation in the CKB Platform contributes to a challenging and competitive educational environment, encouraging students to reach higher levels of proficiency in software development.

## 2.4. Assumptions, dependencies and constraints

The following domain assumptions must hold in the world:

- **D1**: Students are enrolled in the school.
- **D2**: Educators teach in the school.
- **D3**: The school has an existing authentication system that can be used by the CKB platform.
- **D4**: Students and educators have an account on the existing authentication system.
- **D5**: Students correctly fork the GitHub repository of the code kata battle.
- **D6**: Students correctly set up an automated workflow through GitHub Actions.

**D7**: The grade assigned manually by the teachers reflects the work done.

**D8**: The GitHub Action platform is working properly.

**D9**: Test cases and build automation scripts uploaded by educators are correct.

**D10**: The static analysis tool is working.

# 3 | Specific Requirements

## 3.1. External Interface Requirements

### 3.1.1. User Interfaces

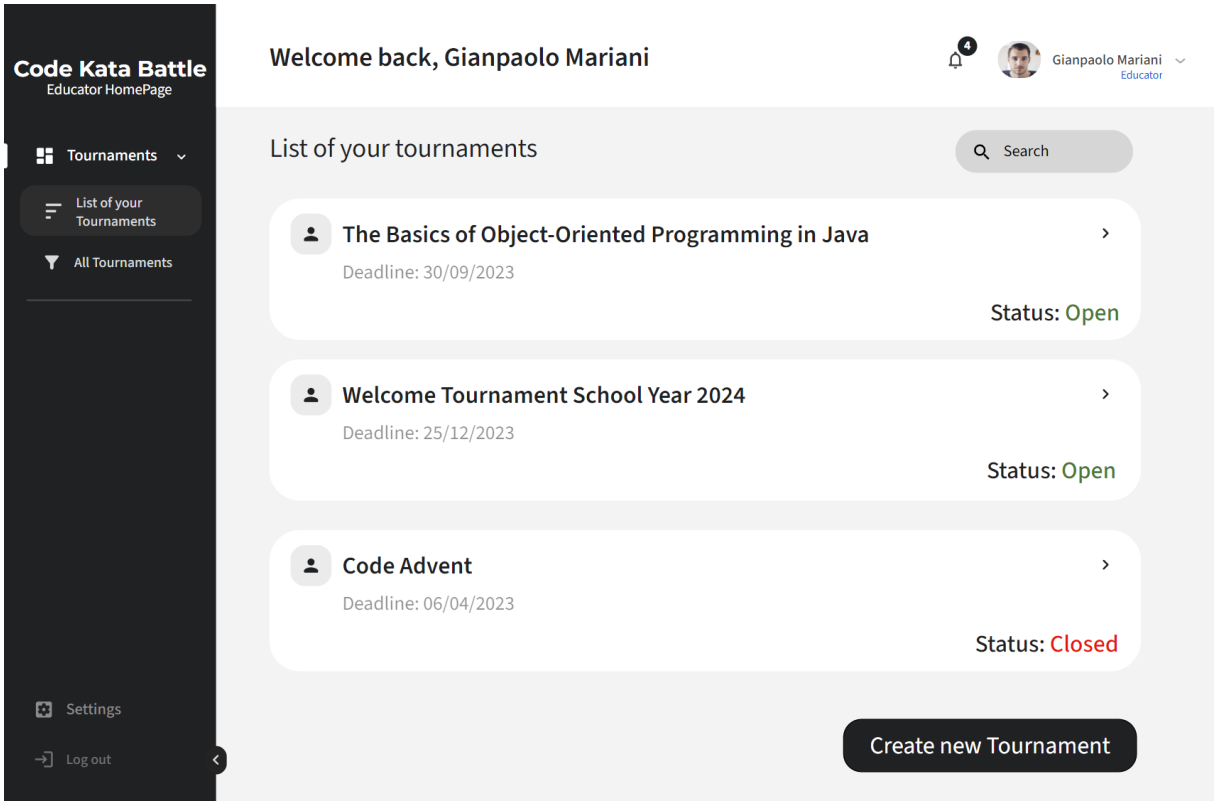



Figure 3.1: HomePage Educator

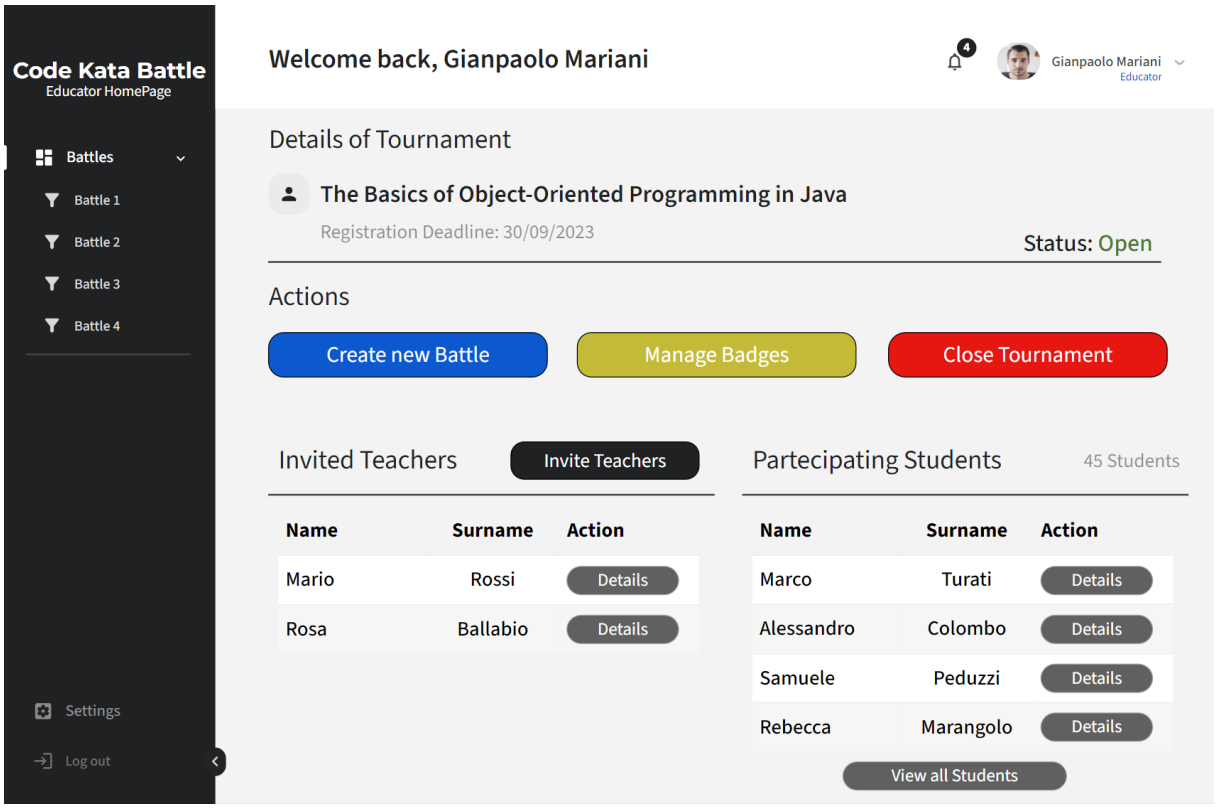



Figure 3.2: Details Tournament Educator

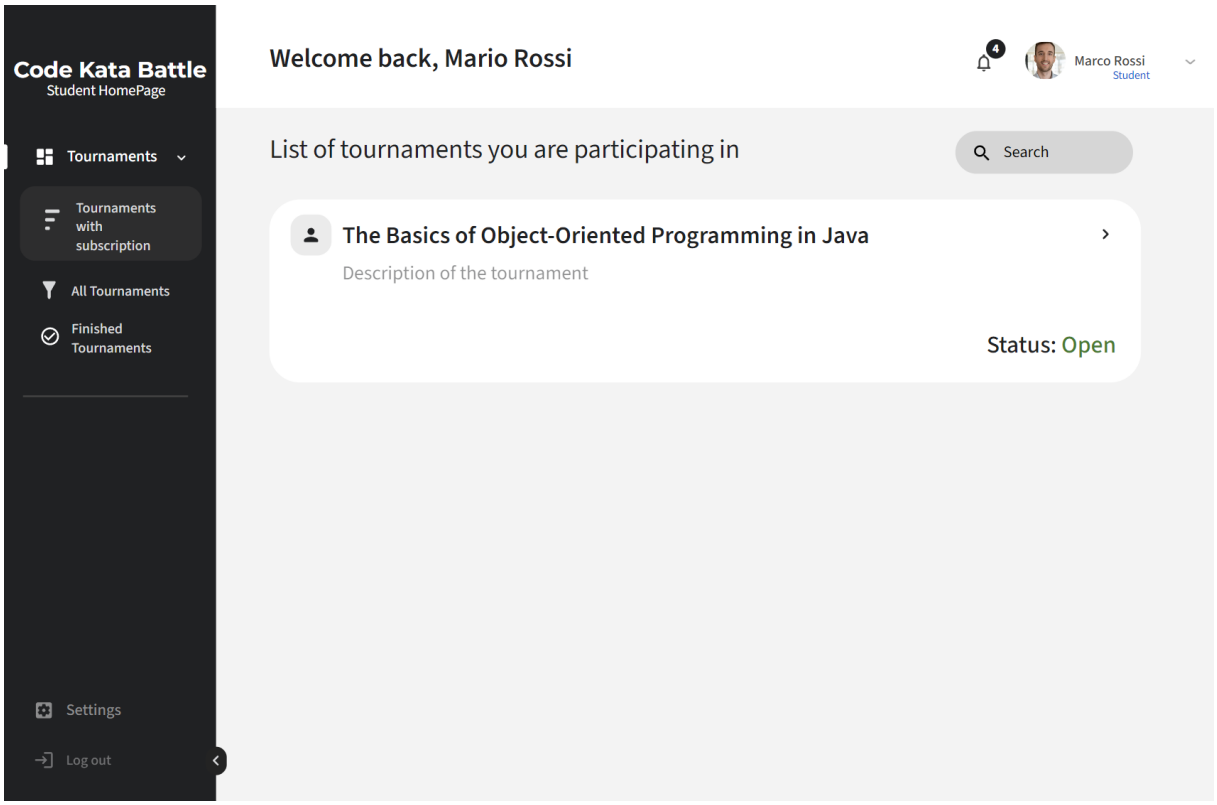



Figure 3.3: HomePage Student

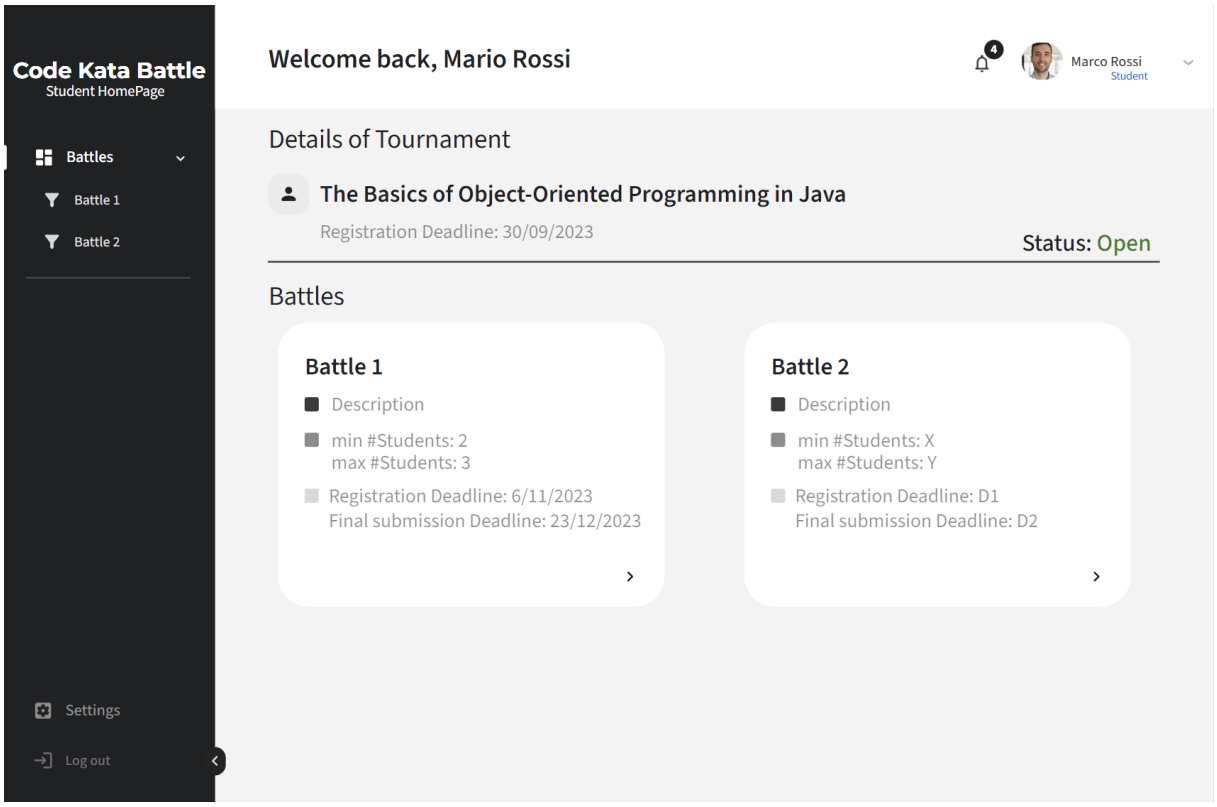



Figure 3.4: Details Tournament Student

### 3.1.2. Hardware Interfaces

The system does not need any specific hardware, each user will need a smartphone or a computer with an Internet connection in order to communicate with the servers of the platform.

### 3.1.3. Software Interfaces

The system will need to interface with the following software:

1. **A git-compatible software [1], [8]** - in order to create git repositories for the battles, push them to GitHub and clone students' repositories in order to evaluate them.

2. **Language-specific software build systems** - in order to be able to build the students projects for evaluation. Different languages have their own requirements for compiling and executing code, so for each supported language the system will need to interact with one or more tools, at minimum with a compiler or interpreter (e.g. GCC or clang for C/C++, javac for Java, node for JavaScript) but more likely with more complex tools (such as make for C/C++, Maven or Gradle for Java, npm for JavaScript).

3. **Language-specific test suites** - in order to evaluate which tests have been passed or failed. Once again, each supported language will need their own implementation, at best by communicating directly with a testing framework (such as JUnit for Java

or Jest for JavaScript) or a build tool (such as Gradle), at worst by directly parsing the text output of tests.

4. **Static analysis tools** - in order to be used for scoring ([12], [9]). In this case, a choice can be made between interfacing with more lower-level software specific tools, which can potentially need per-language support (such as Errorprone, SpotBugs for Java or ESLint for JavaScript), tools which already support a plethora of languages out of the box (such as Infer), or higher level platforms which already interface on their own the lower-level tools (such as SonarQube [10]).

5. **A language interpreter** - to be used to declare variables and evaluate them to be used for the assignment of badges to students. A decision must be made between developing a custom language with its interpreter or the use of an already available interpreted language such as JavaScript or Lua, with the consequent integration needed and the additional constraints to apply in order to disallow potential malicious use.

Additionally, in order to implement the gamification aspect, additional information from each of these tools might be required in order to be able to define variables for badges. For example, we might need to extract more metadata from git related to number of commits by each student, lines changed, the dates of the first/last commit, etc. so that educators are able to use those.

### 3.1.4. Communication Interfaces

The system will need to communicate with the following external platforms:

1. **GitHub** - which is used for code-hosting. The system will need to communicate with GitHub using the git protocol either over HTTP or SSH in order to manage and interact with repositories, but will also need to possibly interact with the GitHub API over HTTP (using their REST API [5] or their GraphQL API [4]) to create new repositories, as well as to collect additional metadata to be used for gamification variables. Additionally, it will also need to receive messages from GitHub workflows, so an API will need to be exposed so that a workflow will be able to notify the system of the push of a new commit by a student

2. **External static analysis platforms** - in case they are chosen to implement static analysis. For example if SonarQube is used, the system will need to be able to communicate using its API to receive reports about projects [11]

3. **3rd party SSO API** - in order to integrate automatically with the universities, to avoid having to manually register each student over, we might need to communicate with their specific Sign On system

Therefore, the following communication protocols are used:

1. **HyperText Transfer Protocol over Secure Socket Layer (HTTPS)** - used for all external communications
2. **Secure Shell (SSH)** - used for git communications. This is to be preferred to HTTPS as the use of SSH certificates will also need to more easily and securely authenticate the system to the GitHub platform [6]
3. **GraphQL [7]** - on top of HTTPS used for communicating with GitHub

## 3.2. Functional Requirements

**R1**: The system shall allow users to log in with SSO.

**R2**: The system shall allow the educator to create a tournament.

- **R2.1**: The system shall allow the educator to specify the subscription deadline of a tournament.
- **R2.2**: The system shall allow the educator to grant other colleagues the permission to create battles within the context of a specific tournament.
- **R2.3**: The system shall notify the students of the new tournament.
- **R2.4**: The system shall allow the educator who create the tournament to close it.

**R3**: The system shall allow students to subscribe to a tournament.

**R4**: The system shall allow the educator to create a battle within the context of a specific tournament.

- **R4.1**: The system shall allow the educator to set minimum and maximum number of students per group.
- **R4.2**: The system shall allow the educator to set a textual description.
- **R4.3**: The system shall allow the educator to set the programming language.
- **R4.4**: The system shall allow the educator to upload a set of test cases.
- **R4.5**: The system shall allow the educator to upload a build automation script.
- **R4.6**: The system shall allow the educator to set a registration deadline.
- **R4.7**: The system shall allow the educator to set a final submission deadline.
- **R4.8**: The system shall allow the educator to enable manual evaluation.
- **R4.9**: The system shall allow the educator to select aspects that should be evaluated by the static analysis tool, such as security, reliability, and maintainability.

**R4.10**: The system shall notify students subscribed to a tournament of the creation of a new battle.

**R5**: The system shall allow students to join a battle, respecting the minimum and maximum number of students per group.

**R5.1**: The system shall allow students to invite other students to join a battle.

**R5.2**: The system shall allow students to accept received invitation.

**R6**: When the registration deadline expires, the system shall create a GitHub repository containing the code kata.

**R7**: The system shall send the link to all students who are members of subscribed teams.

**R8**: The system shall expose an API that can by called by the GitHub Action platform.

**R9**: On each push, the system shall calculate and update the battle score of the team.

**R9.1**: The system shall pull the latest sources.

**R9.2**: The system shall analyze quality level of the sources, based on the aspect selected by the educator.

**R9.3**: The system shall run tests uploaded by the educator.

**R9.4**: The system shall measure the time passed between the registration deadline and the last commit.

**R10**: The system shall allow students and educators involved in the battle to see the current rank evolving during the battle.

**R11**: When the submission deadline expires, if manual evaluation is required, the system shall change the state of the battle to the consolidation stage.

**R12**: When the submission deadline expires, if manual evaluation is not required, the system shall close the battle.

**R13**: During the consolidation stage, if manual evaluation is required, the system shall allow the educator to go through the sources produced by each team to assign his/her score.

**R14**: At the end of a battle, the system shall calculate the final rank.

**R15**: When the final rank is available, the system shall notify all students participating in the battle.

**R16**: At the end of each battle, the platform updates the personal tournament score of each student, that is the sum of all battle scores received in that tournament.

**R17**: The system shall allow users to see the tournament ranks.

**R18**: At the end of a tournament, the system shall calculate the final tournament rank.

**R19**: When the final tournament rank is available, the system shall notify all students involved in the tournament.

**R20**: The system shall allow the educator who create a tournament, to create badges within the context of the tournament.

- **R20.1**: The system shall allow the educator to specify the title of the badge.
- **R20.2**: The system shall allow the educator to create a new variable that represent any piece of information available in the platform relevant for scoring.
- **R20.3**: The system shall allow the educator to specify one or more rules that must be fulfilled to achieve the badge, based on variables.
- **R20.4**: The system shall allow users to visualize badges collected by a student.

**R21**: At the end of a tournament, the system shall assign badges to the student who fulfilled the rules.

### 3.2.1. Use case diagrams

## Student

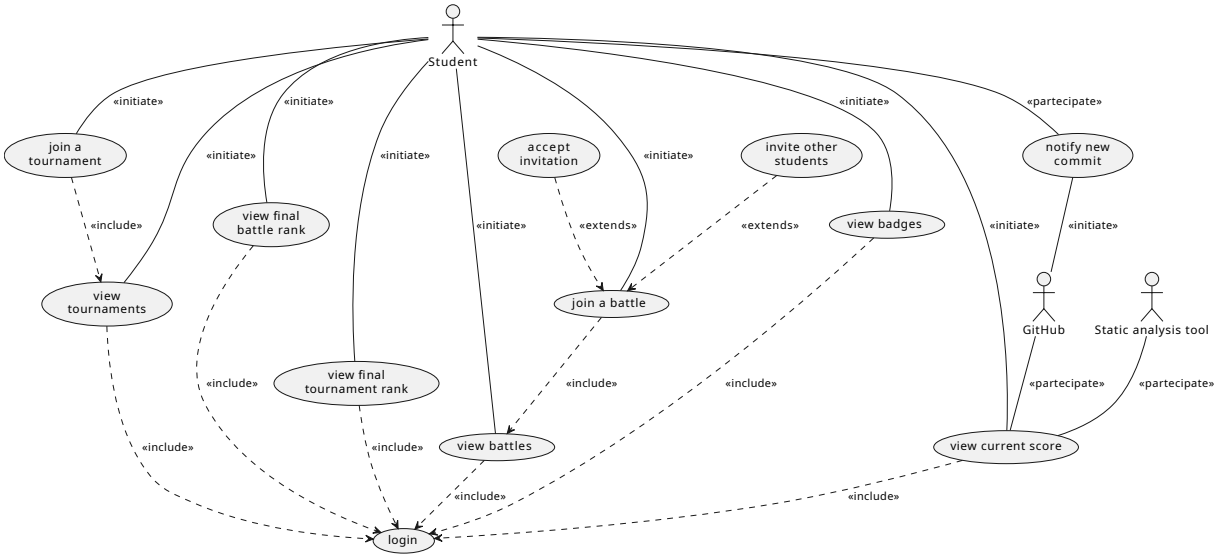



Figure 3.5: Student use case diagram

## Educator

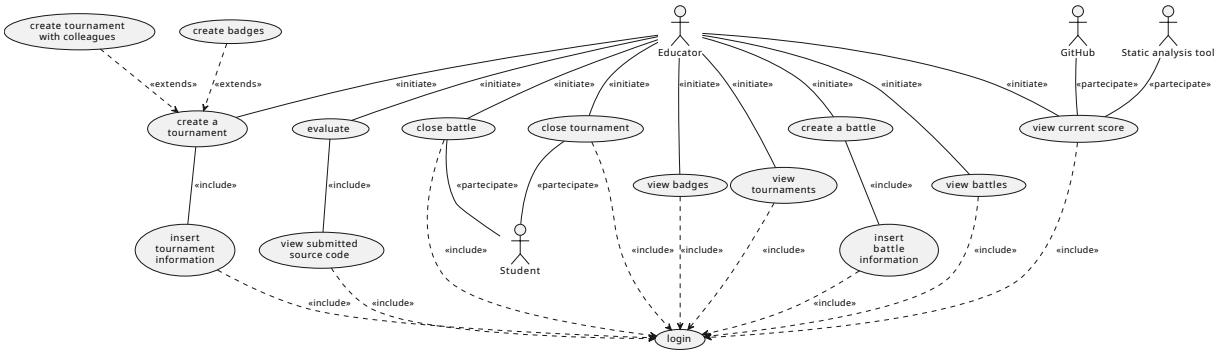



Figure 3.6: Educator use case diagram

### 3.2.2. Use cases

**UC1**:

| | |
|---|---|
| **Name** | Educator creates a new Tournament |
| **Actor** | Educator who wants to create the tournament, Students |
| **Entry condition** | • Educator is logged in |
| **Event flow** | 1. The educator selects the option to create a new tournament<br>2. The educator inserts the name of the tournament.<br>3. The educator inserts the registration deadline of the tournament.<br>4. The educator selects all the other educators he wants to invite to allow them to create new battles for the tournament.<br>5. The educator creates a set of badges.<br>6. The educator confirms the creation of the tournament.<br>7. The platform creates the new tournament.<br>8. The platform sends to the educator a confirmation message.<br>9. The platform sends a notification to all students in the platform. |
| **Exit condition** | The new tournament is created |
| **Exception** | The name of the tournament already exists. In that case, the platform returns an error and does not create the tournament. |

Table 3.1: Educator creates a new Tournament

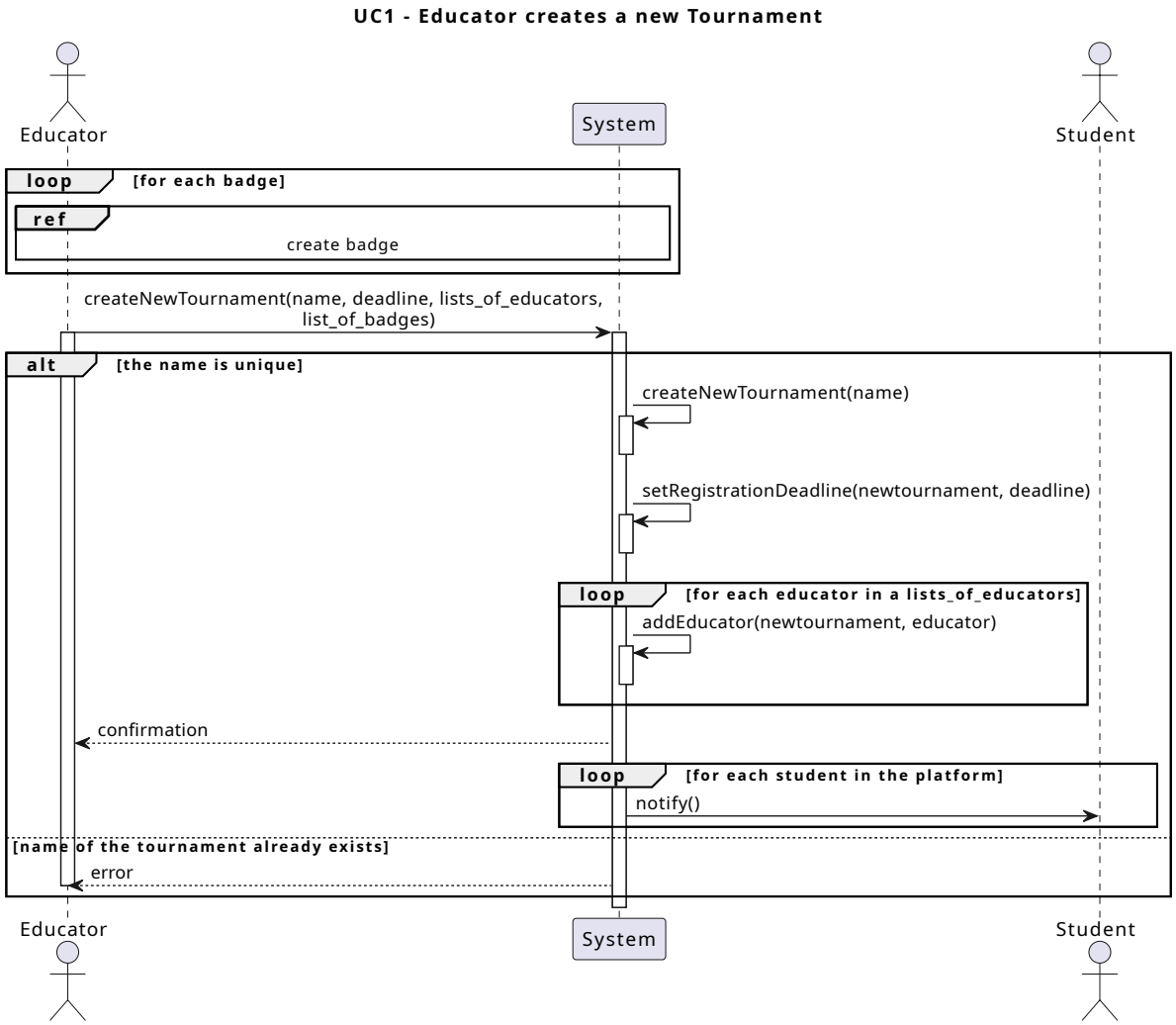



Figure 3.7: Educator creates a new Tournament

**UC2**:

| | |
|---|---|
| **Name** | Educator creates a new badge |
| **Actor** | Educator who wants to create the badge |
| **Entry condition** | • Educator is logged in<br>• Educator is creating a tournament |
| **Event flow** | 1. The educator selects the option to create a new badge.<br>2. The educator inserts the title of the badge.<br>3. The educator writes JavaScript code to create new variables.<br>4. The educator writes one or more rules.<br>5. The educator clicks the confirmation button.<br>6. The platform creates the new badge in the current tournament.<br>7. The platform shows a confirmation massage. |
| **Exit condition** | The new badge is created in the current tournament |
| **Exception** | The rules contain undefined variables. In that case, the platform returns an error and does not create the badge. |

Table 3.2: Educator creates a new badge

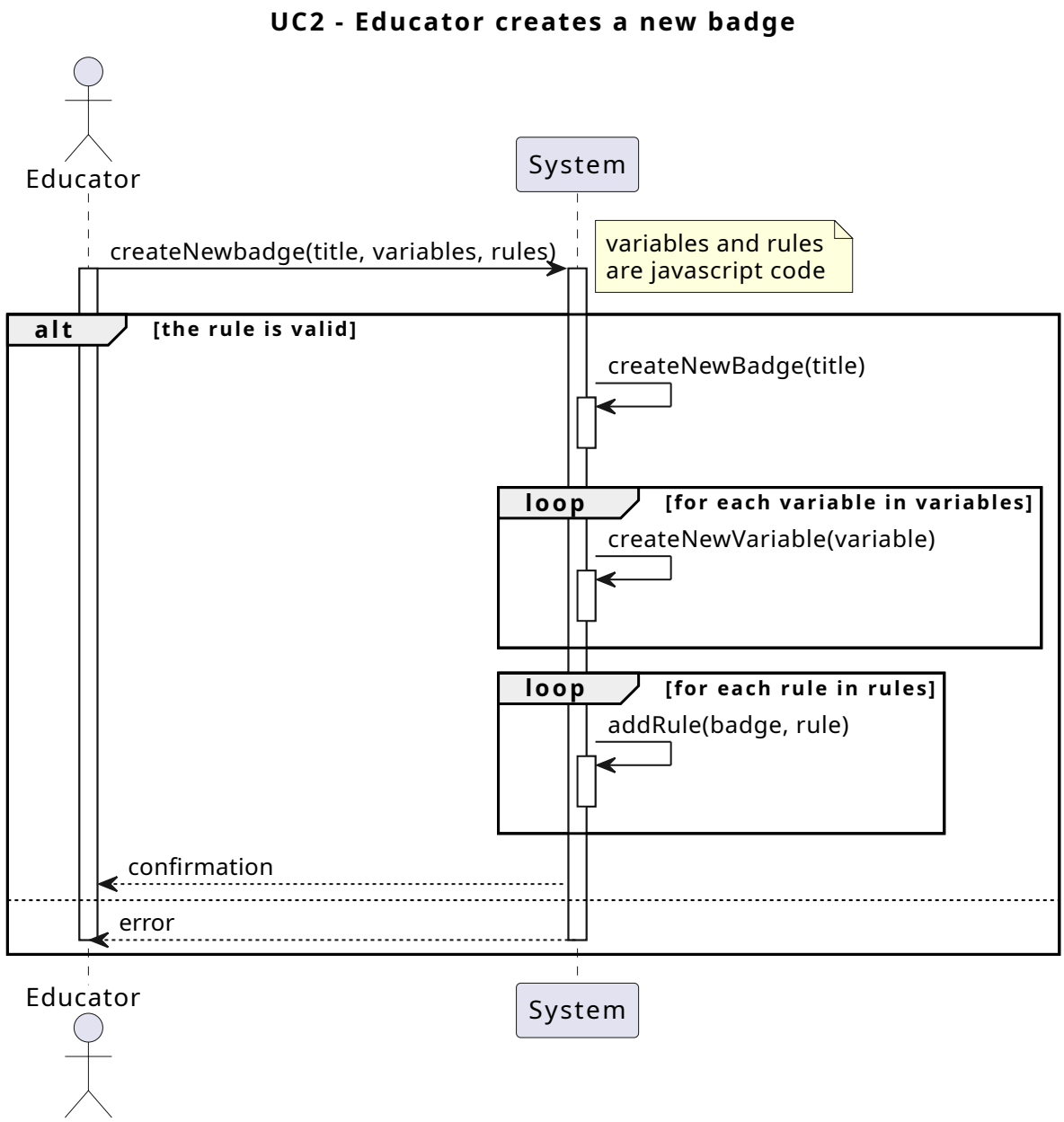



Figure 3.8: Educator creates a new badge

**UC3**:

| | |
|---|---|
| **Name** | Educator creates a new Battle for an Existing Tournament |
| **Actor** | Educator who wants to create the battle, Students participating in the tournament |
| **Entry condition** | • Educator is logged in<br>• Educator has permission to modify the tournament |
| **Event flow** | 1. The educator asks the platform the list of available tournaments.<br>2. The platform sends to the educator the list of available tournaments.<br>3. The educator selects the tournament.<br>4. The platform shows the detail of the selected tournament.<br>5. The educator inserts the Battle Description field.<br>6. The educator uploads test cases and build and automation scripts.<br>7. The educator inserts the minimum and maximum number of students per group.<br>8. The educator inserts the registration deadline.<br>9. The educator inserts the final submission deadline.<br>10. The educator selects the option to create the new battle.<br>11. The platform creates the battle for the tournament.<br>12. The platform sends to the educator a confirmation message.<br>13. The platform sends a notification to all students participating in the tournament. |
| **Exit condition** | The new battle is added to the tournament |
| **Exception** | Some parameters are unfeasible (such as final submission deadline succeed the registration deadline, min number of students > max number of students, ...). In that case, the platform returns an error and does not create the battle. |

Table 3.3: Educator creates a new Battle for an Existing Tournament

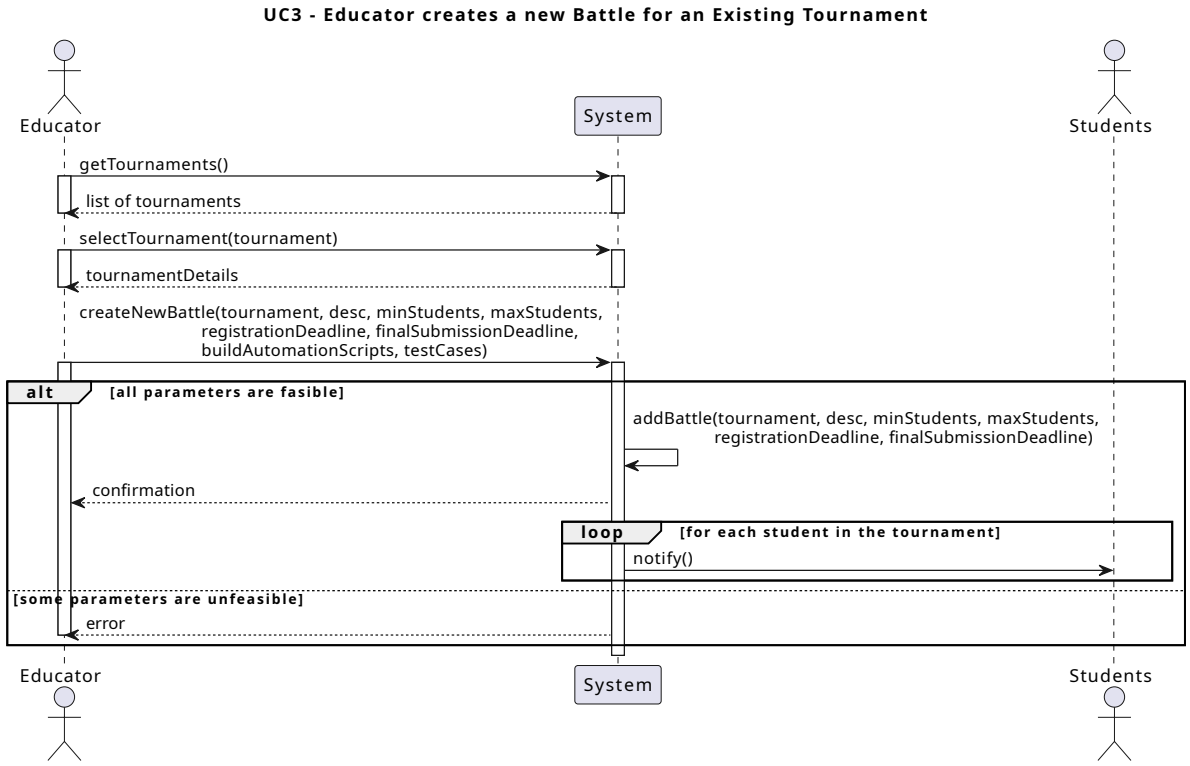



Figure 3.9: Educator creates a new Battle for an Existing Tournament

**UC4**:

| | |
|---|---|
| **Name** | Student joins to an existing Tournament by receiving a notification |
| **Actor** | Student |
| **Entry condition** | • Student is logged in<br>• A new tournament has been created<br>• Student has received a notification of the creation of a new tournament |
| **Event flow** | 1. Upon the creation of a new tournament, the platform sends a notification to all students which have enabled them<br>2. The student asks the platform the list of available tournaments.<br>3. The platform sends to the student the list of available tournaments.<br>4. The student selects the tournament.<br>5. The platform sends to the student the detail of the selected tournament.<br>6. The student selects the option to sign up to the tournament.<br>7. The platform enrolls the student in the tournament.<br>8. The platform sends to the student a confirmation message. |
| **Exit condition** | The student is subscribed to the tournament |
| **Exception** | The student selects the option to sign up to the tournament when the registration deadline has expired. In that case, the platform returns an error. |

Table 3.4: Student joins to an existing Tournament by receiving a notification

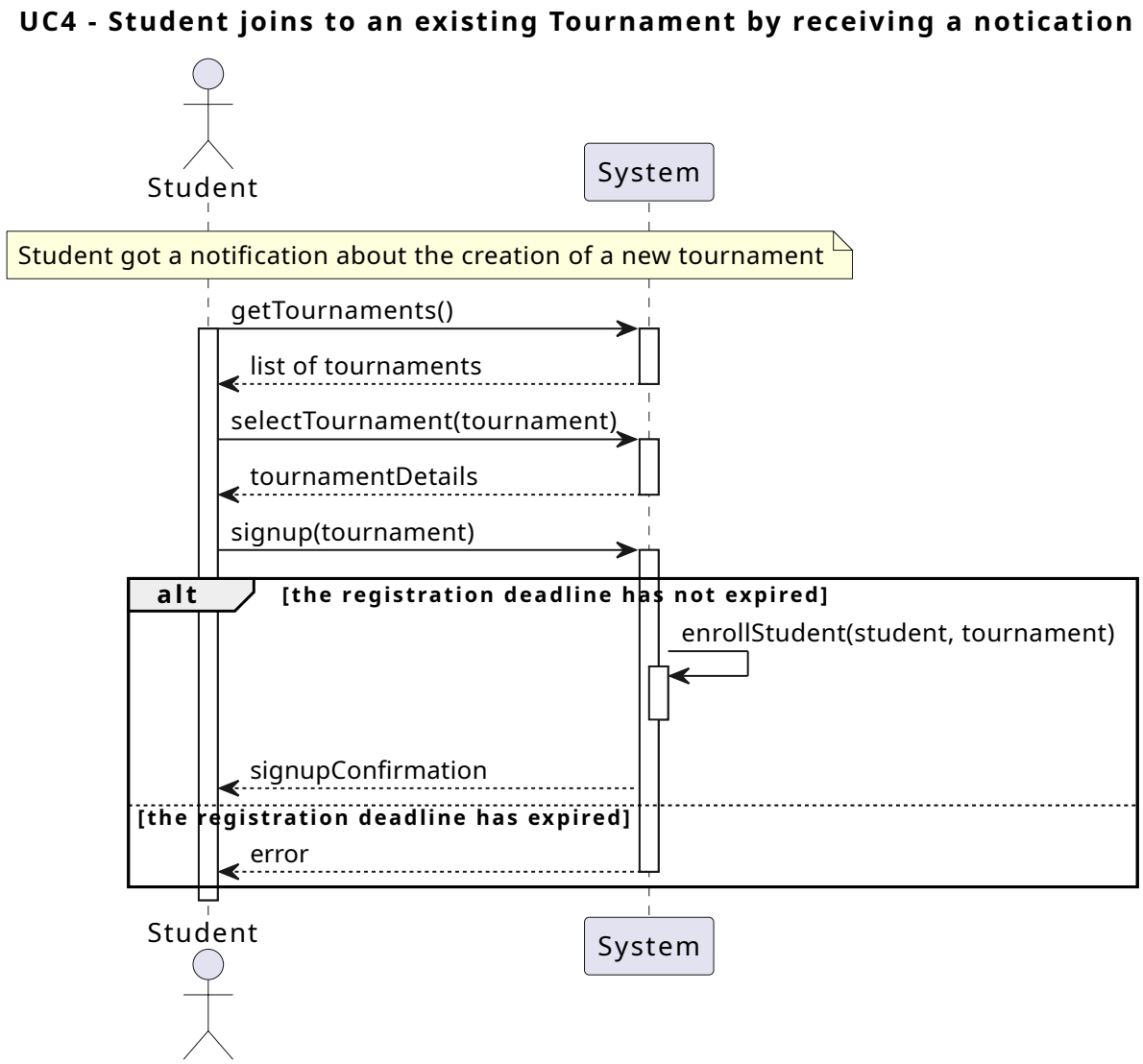



Figure 3.10: Student joins to an existing Tournament by receiving a notification

**UC5**:

| | |
|---|---|
| **Name** | Students create a team for a tournament battle |
| **Actor** | Inviting student, Invited student 1, Invited student 2 |
| **Entry condition** | • Students are logged in<br>• Students are enrolled in the same tournament<br>• Students has invitation notifications enabled |
| **Event flow** | 1. The inviting student asks the platform the list of available tournaments.<br>2. The platform sends to the inviting student the list of available tournaments.<br>3. The inviting student selects the tournament.<br>4. The platform sends to the inviting student the detail of the selected tournament (including the battles available for the tournament).<br>5. The inviting student selects the battle in the tournament.<br>6. The platform sends to the inviting student the detail of the selected battle within the tournament.<br>7. The inviting student selects the option to create a new team for the battle.<br>8. The platform creates a new team for the battle.<br>9. The platform adds the inviting student to the created team.<br>10. The platform returns to the inviting student a confirmation message.<br>11. The inviting student selects the option to invite "invited student 1".<br>12. The platform creates an invitation for invited student 1.<br>13. The platform sends a notification to invited student 1.<br>14. Invited student 1 receives the notification and accepts.<br>15. The platform adds invited student 1 to the team.<br>16. The platform returns to invited student 1 a confirmation message.<br>17. The inviting student selects the option to invite "invited student 2".<br>18. The platform creates an invitation for invited student 2.<br>19. Invited student 2 asks the platform the list of invites. |

| | |
|---|---|
| | 20. The platform returns to Invited student 2 the list of his invites.<br>21. Invited student 2 accepts the invite for the battle.<br>22. The platform adds invited student 2 to the team.<br>23. The platform returns to invited student 2 a confirmation message.<br>24. The inviting student selects the option to participate in the battle with the new created team.<br>25. The platform signs up the team to the battle.<br>26. The platform returns to inviting student a confirmation message. |
| **Exit condition** | The students are all subscribed in a battle with the same team |
| **Exception** | • Inviting student wants to invite another student, but the maximum number of students has been reached. In that case, the platform returns an error.<br>• The team wants to participate in a battle, but the number of participants requirement of the battle is not satisfied. In that case, the platform returns an error and the student can invite additional members. |

Table 3.5: Students create a team for a tournament battle

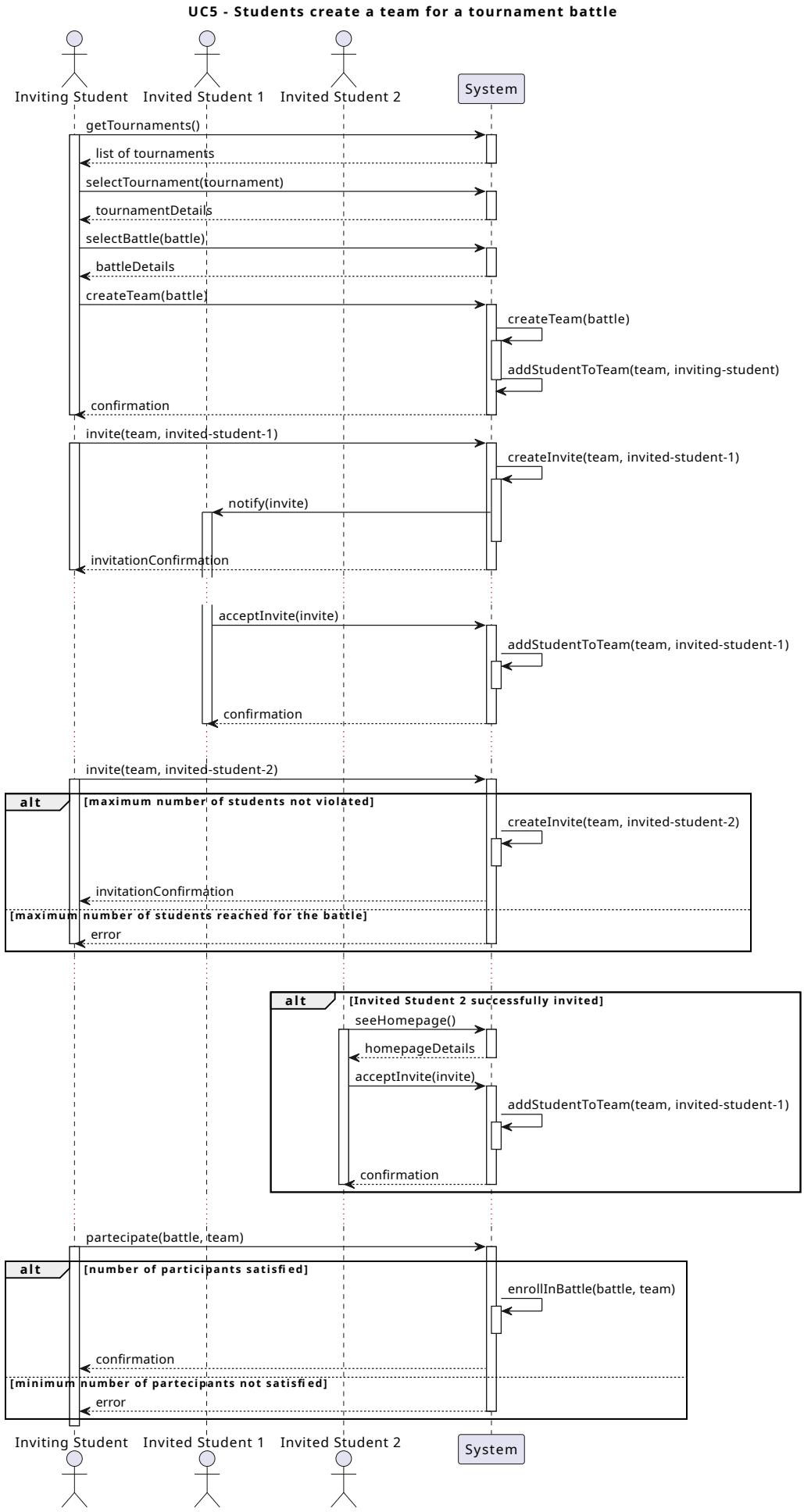



Figure 3.11: Students create a team for a tournament battle

**UC6**:

| | |
|---|---|
| **Name** | Student forks the repository |
| **Actor** | Forking student, teammates, GitHub |
| **Entry condition** | • Students have created a team<br>• The team is subscribed to a battle |
| **Event flow** | 1. When the registration deadline of a battle expires, the platform creates the repository GitHub with name and description of the battle.<br>2. The platform uploads to the repository the build automation scripts and public tests<br>3. The platform notifies all students subscribed to the battle.<br>4. The student forks the repository.<br>5. The student adds his teammates as collaborators in the repository.<br>6. The student enables GitHub Action.<br>7. The student inserts in the platform the link to his repository. |
| **Exit condition** | The GitHub repository for the battle is created |

Table 3.6: Student forks the repository

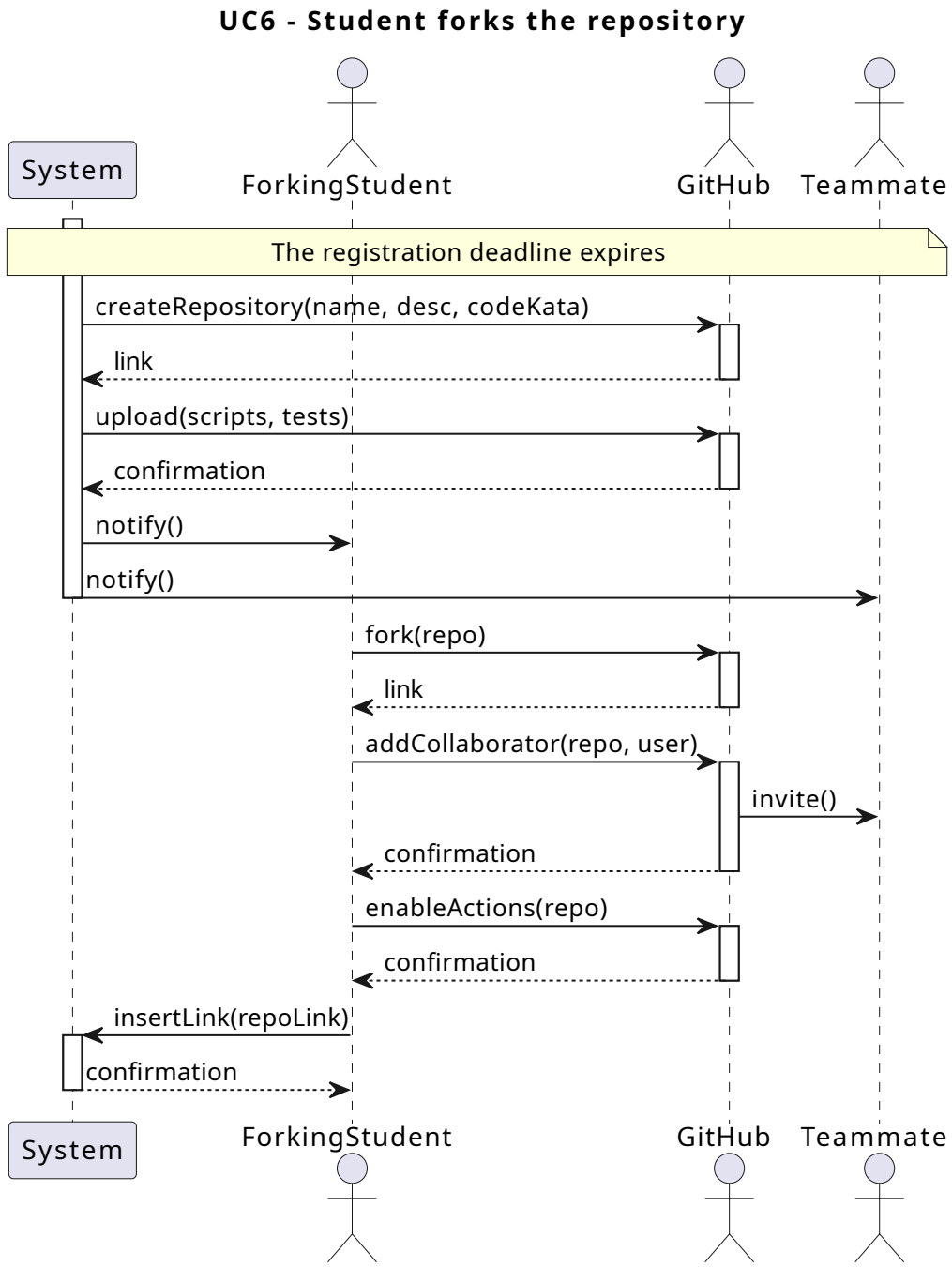



Figure 3.12: Student forks the repository

**UC7**:

| | |
|---|---|
| **Name** | Student pushes and triggers automatic evaluation |
| **Actor** | Pushing Student, Students participating in the same team, GitHub, Static analysis tool |
| **Entry condition** | • Pushing Student is participating in a battle of a tournament<br>• Pushing Student has forked the GitHub repository of the battle<br>• Pushing Student has set up the GitHub action on the forked repository<br>• Pushing Student has inserted the repository link in the platform |
| **Event flow** | 1. The pushing Student commits project code.<br>2. The pushing Student pushes the committed project code to the forked GitHub battle repository.<br>3. The GitHub Action platform notifies the CKB platform about the new push.<br>4. The platform pulls the updated project code.<br>5. The platform builds the project.<br>6. The platform runs all the test cases (public and private) associated with the battle.<br>7. The platform calculates the days that have passed since the start of the battle.<br>8. The platform runs the static analysis tool.<br>9. The platform calculates the total score.<br>10. The platform updates the battle team score.<br>11. The platform notifies the updated battle team score to all the students of the same team. |
| **Exit condition** | The battle team score is updated |
| **Exception** | The platform receives a push notification from an unauthorized repository. In this case the request is discarded. |
| **Special reqs** | The platform must evaluate the project within 10 minutes of the push. |

Table 3.7: Student pushes and triggers automatic evaluation

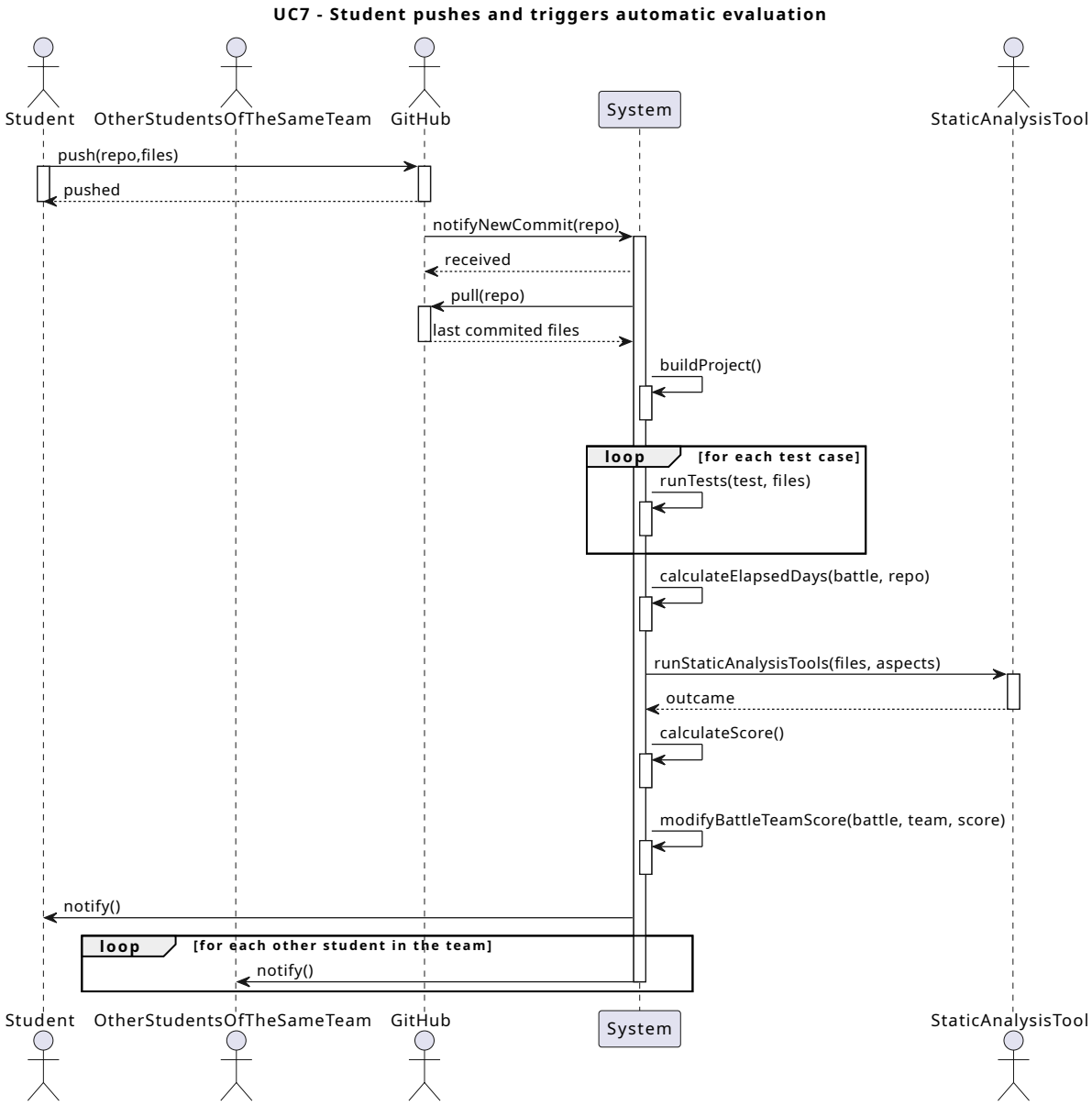



Figure 3.13: Student pushes and triggers automatic evaluation

**UC8**:

| | |
|---|---|
| **Name** | Educator manually evaluates teams |
| **Actor** | Educator, Students, GitHub |
| **Entry condition** | • Educator has logged in<br>• Educator has permission to modify the battle of the tournament<br>• Educator has enables manual evaluation for the battle |
| **Event flow** | 1. When the submission deadline expires, the platform opens the consolidation stage.<br>2. The educator asks the platform the list of available tournaments.<br>3. The platform sends to the educator the list of available tournaments.<br>4. The educator selects the tournament.<br>5. The platform shows the detail of the selected tournament (including the list of battles).<br>6. The educator selects the battle.<br>7. The platform shows the details of the battle.<br>8. The educator selects the option to evaluate teams.<br>9. The platform returns the list of teams within the battle.<br>10. The educator asks for the source code for each team.<br>11. The platform returns to the educator the source code for each team.<br>12. The educator assigns a score to each team.<br>13. The educator selects the option to close the battle.<br>14. The platform calculates the final battle ranks.<br>15. The platform updates the personal tournament score of each student.<br>16. The platform notifies students who participated in the battle.<br>17. The platform sends to the educator a confirmation message. |
| **Exit condition** | The battle is closed, the final battle ranks are calculated and the personal tournament scores are updated |
| **Exception** | The educator selects the option to close the battle before evaluating all students. In this case the platform returns an error and the educator can assign the missing grades. |

Table 3.8: Educator manually evaluates teams

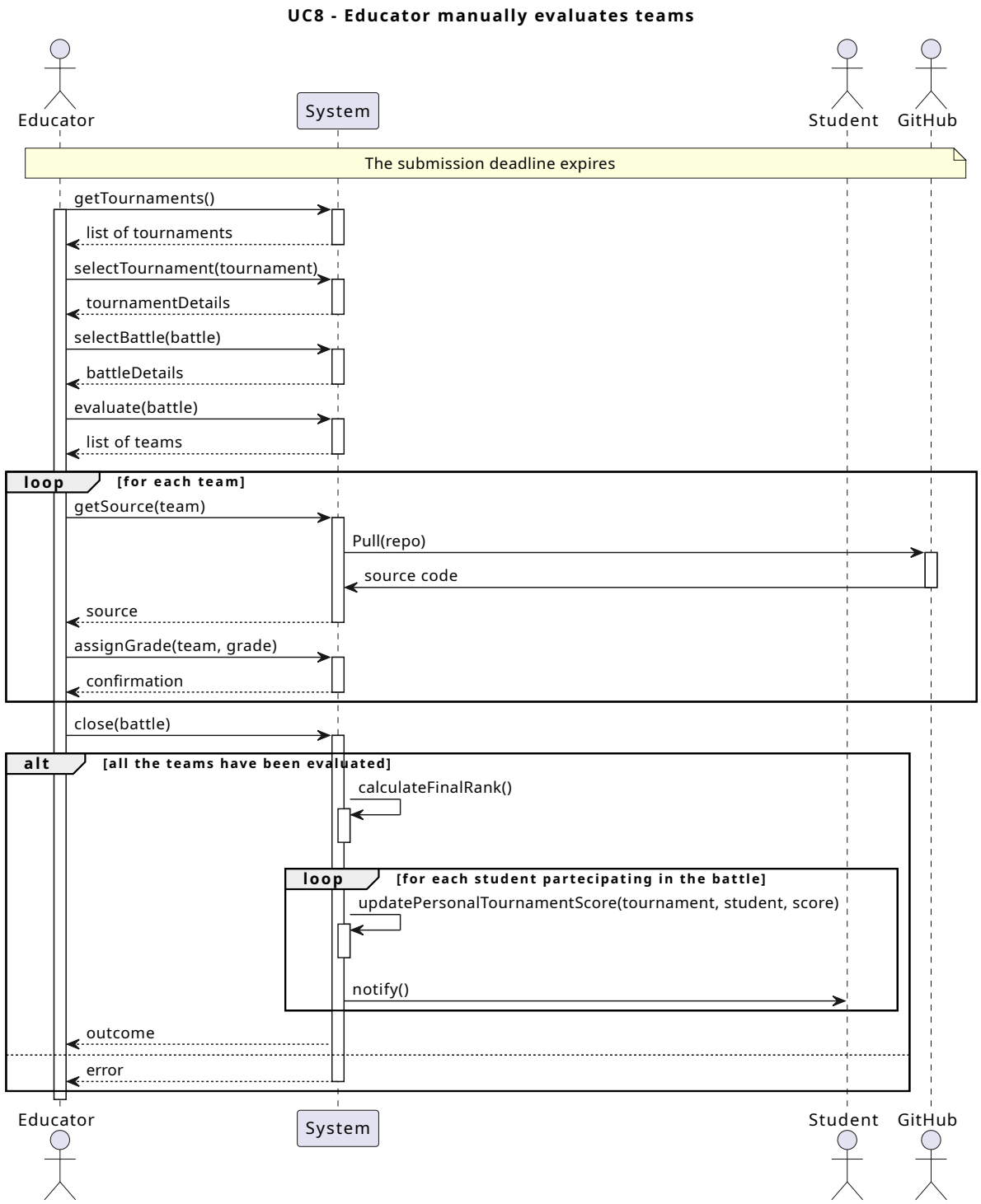



Figure 3.14: Educator manually evaluates teams

**UC9**:

| | |
|---|---|
| **Name** | Educator closes a tournament |
| **Actor** | Educator, Students |
| **Entry condition** | • Educator has logged in<br>• Educator has permission to modify the tournament<br>• All battles in the tournament are closed |
| **Event flow** | 1. The educator asks the platform the list of available tournaments.<br>2. The platform sends to the educator the list of available tournaments.<br>3. The educator selects the tournament.<br>4. The platform shows the detail of the selected tournament.<br>5. Educator selects the option to close the tournament.<br>6. The platform calculates the final tournament rank.<br>7. The platform gets the information about the available badges for the tournament.<br>8. The platform assigns badges to the student who fulfilled the rules.<br>9. The platform notifies students who participated in the tournament. |
| **Exit condition** | The tournament is closed and the badges are assigned |

Table 3.9: Educator closes a tournament

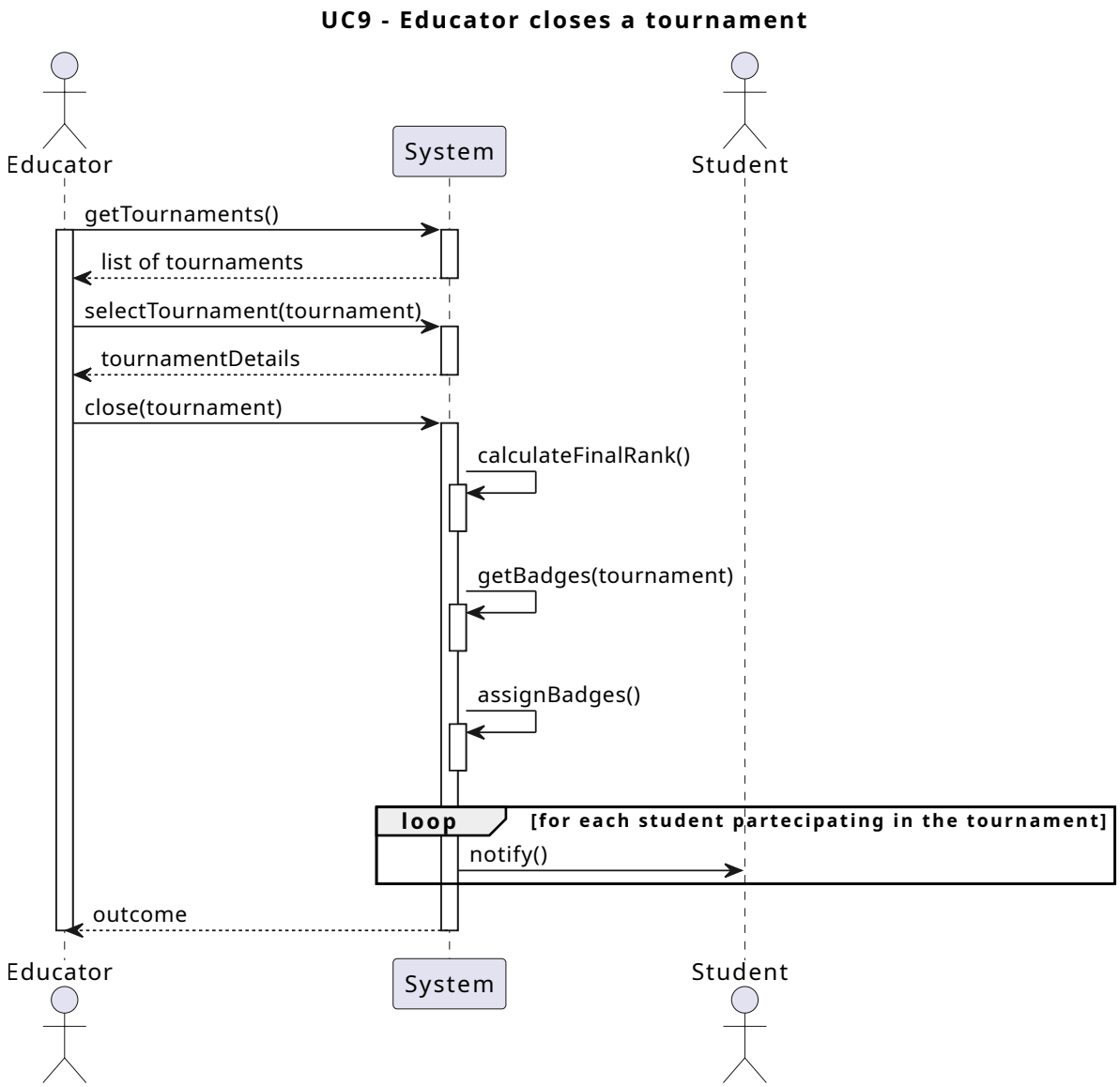



Figure 3.15: Educator closes a tournament

### 3.2.3. Requirements mapping

In this section it is shown how the relation $R \wedge D \models G$ holds. In particular, the following traceability matrix associates domain assumptions and requirements to goals. After that, to facilitate reading, the text of all the assumptions and all the requirements related to each goal is reported.

| Goal ID | Requirement ID | Domain assumption ID |
|---|---|---|
| G1 | R1, R2, R2.1, R2.4 | D1, D2, D3, D4 |
| G2 | R1, R4, R4.1, R4.2, R4.3, R4.4, R4.5, R4.6, R4.7 | D2, D3, D4 |
| G3 | R1, R2.2 | D2, D3, D4 |
| G4 | R1, R4.8, R4.9 | D2, D3, D4 |
| G5 | R1, R2.3 | D1, D2, D3, D4 |
| G6 | R1, R3 | D1, D3, D4 |
| G7 | R1, R4.10 | D1, D3, D4 |
| G8 | R1, R5.1, R5.2 | D1, D3, D4 |
| G9 | R1, R5, R6, R7 | D1, D3, D4, D5 |
| G10 | R1, R8, R9, R9.1, R9.2, R9.3, R9.4, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19 | D1, D3, D4, D5, D6, D7, D8, D9, D10 |
| G11 | R1, R20, R20.1, R20.2, R20.3 | D2, D3, D4 |
| G12 | R1, R21 | D1, D3, D4 |
| G13 | R1, R20.4 | D1, D2, D3, D4 |

Table 3.10: Traceability matrix

**G1**: Educators can organize tournaments of programming exercises aimed at improving their students' skills.

**R1**: The system shall allow users to log in with SSO.

**R2**: The system shall allow the educator to create a tournament.

**R2.1**: The system shall allow the educator to specify the subscription deadline of a tournament.

**R2.4**: The system shall allow the educator who create the tournament to close it.

**D1**: Students are enrolled in the school.

**D2**: Educators teach in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication

system.

**G2**: Educators can organize battles in a tournament they have created.

**R1**: The system shall allow users to log in with SSO.

**R4**: The system shall allow the educator to create a battle within the context of a specific tournament.

**R4.1**: The system shall allow the educator to set minimum and maximum number of students per group.

**R4.2**: The system shall allow the educator to set a textual description.

**R4.3**: The system shall allow the educator to set the programming language.

**R4.4**: The system shall allow the educator to upload a set of test cases.

**R4.5**: The system shall allow the educator to upload a build automation script.

**R4.6**: The system shall allow the educator to set a registration deadline.

**R4.7**: The system shall allow the educator to set a final submission deadline.

**D2**: Educators teach in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**G3**: Educators can aid their colleagues in creating battles.

**R1**: The system shall allow users to log in with SSO.

**R2.2**: The system shall allow the educator to grant other colleagues the permission to create battles within the context of a specific tournament.

**D2**: Educators teach in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**G4**: Educator can decide how to score a battle they have created.

**R1**: The system shall allow users to log in with SSO.

**R4.8**: The system shall allow the educator to enable manual evaluation.

**R4.9**: The system shall allow the educator to select aspects that should be evaluated by the static analysis tool, such as security, reliability, and maintainability.

**D2**: Educators teach in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**G5**: Students can be notified about new tournaments.

**R1**: The system shall allow users to log in with SSO.

**R2.3**: The system shall notify the students of the new tournament.

**D1**: Students are enrolled in the school.

**D2**: Educators teach in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**G6**: Students can subscribe to tournaments.

**R1**: The system shall allow users to log in with SSO.

**R3**: The system shall allow students to subscribe to a tournament.

**D1**: Students are enrolled in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**G7**: Students can be notified about new battles in tournaments they are subscribed to.

**R1**: The system shall allow users to log in with SSO.

**R4.10**: The system shall notify students subscribed to a tournament of the creation of a new battle.

**D1**: Students are enrolled in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**G8**: Students can form teams to take part in a battle.

**R1**: The system shall allow users to log in with SSO.

**R5.1**: The system shall allow students to invite other students to join a battle.

**R5.2**: The system shall allow students to accept received invitation.

**D1**: Students are enrolled in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**G9**: Teams of students can join a battle.

**R1**: The system shall allow users to log in with SSO.

**R5**: The system shall allow students to join a battle, respecting the minimum and maximum number of students per group.

**R6**: When the registration deadline expires, the system shall create a GitHub repository containing the code kata.

**R7**: The system shall send the link to all students who are members of subscribed teams.

**D1**: Students are enrolled in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**D5**: Students correctly fork the GitHub repository of the code kata battle.

**G10**: Students and educators can monitor the progress of all students in battles and tournaments.

**R1**: The system shall allow users to log in with SSO.

**R8**: The system shall expose an API that can by called by the GitHub Action platform.

**R9**: On each push, the system shall calculate and update the battle score of the team.

**R9.1**: The system shall pull the latest sources.

**R9.2**: The system shall analyze quality level of the sources, based on the aspect selected by the educator.

**R9.3**: The system shall run tests uploaded by the educator.

**R9.4**: The system shall measure the time passed between the registration deadline and the last commit.

**R10**: The system shall allow students and educators involved in the battle to see the current rank evolving during the battle.

**R11**: When the submission deadline expires, if manual evaluation is required, the system shall change the state of the battle to the consolidation stage.

**R12**: When the submission deadline expires, if manual evaluation is not required, the system shall close the battle.

**R13**: During the consolidation stage, if manual evaluation is required, the system shall allow the educator to go through the sources produced by each team to assign his/her score.

**R14**: At the end of a battle, the system shall calculate the final rank.

**R15**: When the final rank is available, the system shall notify all students participating in the battle.

**R16**: At the end of each battle, the platform updates the personal tournament score of each student, that is the sum of all battle scores received in that tournament.

**R17**: The system shall allow users to see the tournament ranks.

**R18**: At the end of a tournament, the system shall calculate the final tournament rank.

**R19**: When the final tournament rank is available, the system shall notify all students involved in the tournament.

**D1**: Students are enrolled in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**D5**: Students correctly fork the GitHub repository of the code kata battle.

**D6**: Students correctly set up an automated workflow through GitHub Actions.

**D7**: The grade assigned manually by the teachers reflects the work done.

**D8**: The GitHub Action platform is working properly.

**D9**: Test cases and build automation scripts uploaded by educators are correct.

**D10**: The static analysis tool is working.

**G11**: Educators can manage students' gamification badges.

**R1**: The system shall allow users to log in with SSO.

**R20**: The system shall allow the educator who create a tournament, to create badges within the context of the tournament.

**R20.1**: The system shall allow the educator to specify the title of the badge.

**R20.2**: The system shall allow the educator to create a new variable that represent any piece of information available in the platform relevant for scoring.

**R20.3**: The system shall allow the educator to specify one or more rules that must be fulfilled to achieve the badge, based on variables.

**D2**: Educators teach in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**G12**: Students can receive badges depending on the rules they have fulfilled.

**R1**: The system shall allow users to log in with SSO.

**R21**: At the end of a tournament, the system shall assign badges to the student who fulfilled the rules.

**D1**: Students are enrolled in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

**G13**: Students and educators can see badges collected by a student.

**R1**: The system shall allow users to log in with SSO.

**R20.4**: The system shall allow users to visualize badges collected by a student.

**D1**: Students are enrolled in the school.

**D2**: Educators teach in the school.

**D3**: The school has an existing authentication system that can be used by the CKB platform.

**D4**: Students and educators have an account on the existing authentication system.

The following traceability matrix associates requirements to use cases.

| | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 | UC8 | UC9 |
|---|---|---|---|---|---|---|---|---|---|
| **R1** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **R2** | ✓ | | | | | | | | |
| **R2.1** | ✓ | | | | | | | | |
| **R2.2** | ✓ | | | | | | | | |
| **R2.3** | ✓ | | | ✓ | | | | | |
| **R2.4** | | | | | | | | | ✓ |
| **R3** | | | | ✓ | | | | | |
| **R4** | | | ✓ | | | | | | |
| **R4.1** | | | ✓ | | | | | | |
| **R4.2** | | | ✓ | | | | | | |
| **R4.3** | | | ✓ | | | | | | |
| **R4.4** | | | ✓ | | | | | | |
| **R4.5** | | | ✓ | | | | | | |
| **R4.6** | | | ✓ | | | | | | |
| **R4.7** | | | ✓ | | | | | | |
| **R4.8** | | | ✓ | | | | | | |
| **R4.9** | | | ✓ | | | | | | |
| **R4.10** | | | ✓ | | | | | | |
| **R5** | | | | | ✓ | | | | |
| **R5.1** | | | | | ✓ | | | | |
| **R5.2** | | | | | ✓ | | | | |
| **R6** | | | | | | ✓ | | | |
| **R7** | | | | | | ✓ | | | |
| **R8** | | | | | | | ✓ | | |
| **R9** | | | | | | | ✓ | | |
| **R9.1** | | | | | | | ✓ | | |
| **R9.2** | | | | | | | ✓ | | |
| **R9.3** | | | | | | | ✓ | | |
| **R9.4** | | | | | | | ✓ | | |
| **R10** | | | | | | | ✓ | | |
| **R11** | | | | | | | | ✓ | |
| **R12** | | | | | | | | ✓ | |
| **R13** | | | | | | | | ✓ | |
| **R14** | | | | | | | | ✓ | |
| **R15** | | | | | | | | ✓ | |
| **R16** | | | | | | | | ✓ | |
| **R17** | | | | | | | | | ✓ |

| | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 | UC8 | UC9 |
|---|---|---|---|---|---|---|---|---|---|
| **R18** | | | | | | | | | ✓ |
| **R19** | | | | | | | | | ✓ |
| **R20** | | ✓ | | | | | | | |
| **R20.1** | | ✓ | | | | | | | |
| **R20.2** | | ✓ | | | | | | | |
| **R20.3** | | ✓ | | | | | | | |
| **R20.4** | | | | | | | | | ✓ |
| **R21** | | | | | | | | | ✓ |

Table 3.11: Use cases traceability matrix

## 3.3. Performance Requirements

The system must be sized considering the number of users who will use it. Indeed, the grade assigned by the platform must not depend on the number of students pushing at the same time.

The automatic evaluation must be done within 10 minutes of the push. If this does not happen, the platform can assume that the tests are deadlocked and kill the process.

The system must have disk space to store information of tournaments and battles, such as teams and ranks. Students' code is hosted on GitHub, but it has to be cloned to the server disk to run tests.

## 3.4. Design Constraints

### 3.4.1. Standards Compliance

The CodeKataBattle (CKB) platform is designed with the goal of strictly adhering to several standards to ensure the quality, security and interoperability of the system.
The standards to which CKB adheres include:

1. HTTPS Protocol Compliance
   Implementation of the HTTPS protocol according to the cryptographic standards established by IETF (Internet Engineering Task Force).

2. Accessibility Standards
   Compliance with Web Content Accessibility Guidelines (WCAG) to ensure web accessibility.

3. Security Standards
   Implementation of security best practices as defined by OWASP (Open Web Application Security Project) and NIST (National Institute of Standards and Technology). Such as password storage encryption with HASH512 + Salt, SSL Certificate

and end-to-end communication encryption.

4. API Interoperability
   Use of open standards for API design, such as RESTful API and adherence to specifications such as OpenAPI (Swagger).

5. Coding Standards
   The platform follows universally accepted coding guidelines for the main programming language used in system development (e.g., Java or Python). Compliance with the coding guidelines for the main programming language (PEP 8 for Python, Java Coding Conventions for Java).

6. Compliance with Privacy Regulations
   Compliance with privacy laws such as the General Data Protection Regulation (GDPR) for European citizens.

### 3.4.2. Hardware limitations

Each student and educator must have an electronic device such as a computer or smartphone with an internet connection to access the CKB Platform. Students must be able to clone GitHub repositories on their device to participate in the various battles and accordingly must have a physical storage with sufficient space to store the sources and data of the various projects. The device used must allow the student to view, edit, process and execute the sources. The devices used by students and educators enable them to receive notifications.

## 3.5. Software System Attributes

### 3.5.1. Reliability

The CKB platform does not manage critical operations. If an operation fails, it can be re-executed without any particular consequences. For example, if the evaluation of an exercise fails, students can resubmit their code. It is therefore reasonable to have a failure rate around 1%.

### 3.5.2. Availability

The platform should have the lowest downtime possible during the day. A few hours of downtime are allowed between 1am and 5am, when students are usually not studying. The platform shall therefore guarantee 99% (two-nines) of availability, so that 3.65 days of downtime per year are allowed.

### 3.5.3. Security

Communication between the user and the CBK platform is encrypted. Furthermore, users must only be able to perform operations that they are authorized to do. For example, a student must not be able to change his grade.

The CKB platform can be triggered through its API only from authorized repositories.

### 3.5.4. Maintainability

The system should be divided in scalable and reusable modules, which will be easier to maintain and replace in case of failure. Ordinary maintenance, for bug fixes and improvements, will be scheduled during night time, when the user traffic is minimal.

### 3.5.5. Portability

The CKD platform does not require any particular hardware or software. It must be accessible from any operating system with a modern web browser. A mobile application that allows users to see the state of the battles can also be developed. Since the mobile app does not require any special function, a non-native approach can be adopted. It is therefore possible to use cross-platform development tools, which can speed up the development process.

# 4 | Formal analysis using alloy

This section describes the entire model formally using the alloy language. In particular, we aim to:

1. model the interaction between students in order to create teams which can participate to battles
2. model the interactions related to the creation and management of tournaments as well as battles, in order to prove that authorization is represented properly by the model

## 4.1. Alloy Code

```
open util/ordering[DateTime]

sig DateTime {}

sig Educator {
        , createdTournaments: set Tournament
        , allowedTournaments: set Tournament
        , battles: set Battle
        , badges: set Badge
} {
        // Sync the two relationships
        all t: Tournament | t in createdTournaments  ⇔  t.creator = this
        // Sync the two relationships
        all t: Tournament | t in allowedTournaments  ⇔  this in t.authorized
        // Sync the two relationships
        all b: Battle | b in battles  ⇔  b.creator = this
        // Sync the two relationships
        all b: Badge | b in badges  ⇔  b.creator = this
}

sig Badge {
        , creator: Educator
        , tournaments: set Tournament
        , title: String
        , condition: String
        , earners: set TournamentSubscription
} {
        title = "Title"
        condition = "predicate"
        // Sync the two relationships
        all t: Tournament | t in tournaments  ⇔  this in t.badges
        // Sync the two relationships
        all ts: TournamentSubscription | ts in earners  ⇔  this in ts.badges
}

sig Tournament {
        , creator: Educator
        , authorized: set Educator
        , state: TournamentState
        , subscriptionDeadline: DateTime
```

```
        , battles: set Battle
        , subscriptions: set TournamentSubscription
        , badges: set Badge
} {
        // Sync the two relationships
        all b: Battle | (b in battles) ⇔ (b.tournament = this)
        // Sync the two relationships
        all s: TournamentSubscription | (s in subscriptions) ⇔ (s.tournament = this)
}

enum TournamentState { Subscribing, InProgress, TournamentDone }

sig Battle {
        , creator: Educator
        , tournament: Tournament
        , state: BattleState
        , description: String
        , minStudents: Int
        , maxStudents: Int
        , registrationDeadline: DateTime
        , submissionDeadline: DateTime
        , participations: set BattleParticipation
        , repo: lone Repository
} {
        description = "descr"
        // Min and max students constraints
        minStudents > 0 and minStudents ≤ maxStudents
        // Sync the two relationships
        all p: BattleParticipation | (p in participations) ⇔ (p.battle = this)
        // Repo must exist after the registration state
        gt[state, Registration] ⇔ repo ≠ none
        // creator must be tournament creator or authorized
        creator = tournament.creator or creator in tournament.authorized
        // Sync the two relationships
        repo ≠ none ⇔ repo.battle = this
}

enum BattleState { Registration, Submission, Consolidation, BattleDone }

sig Student {
        , receivedInvites: set Invite
        , sentInvites: set Invite
        , subscriptions: set TournamentSubscription
        , participations: set TeamParticipation
} {
        // Sync the two relationships
        all i: Invite | (i in receivedInvites) ⇔ i.invitee = this
        // Sync the two relationships
        all i: Invite | (i in sentInvites) ⇔ i.inviter = this
        // Sync the two relationships
        all s: TournamentSubscription | (s in subscriptions) ⇔ s.student = this
        // Sync the two relationships
        all p: TeamParticipation | (p in participations) ⇔ p.student = this
}

sig TournamentSubscription {
        , student: Student
        , tournament: Tournament
        , subscriptionDate: DateTime
        , score: Int
        , badges: set Badge
} {
        // Must be subscribed before the deadline
```

```
        lte[subscriptionDate, tournament.subscriptionDeadline]
        // If the tournament has not started yet, can't have badges
        lte[tournament.state, Subscribing] ⟹ not (some b: Badge | b in badges)
        // Earned badge must be defined for this tournament
        all b: Badge | b in badges ⟹ b in tournament.badges
}

sig Invite {
        , state: InviteState
        , inviter: Student
        , invitee: Student
        , teamParticipation: TeamParticipation
} {
        // A student can't invite itself
        inviter ≠ invitee
        // If the student has accepted, he must have a team
        state = Accepted implies (some t: TeamParticipation | t.invite = this)
        // The invited student must be the same participating
        teamParticipation.student = invitee
}

fact {
        // There cannot be two pending/accepted invites for the same team
        all disj i1, i2: Invite | (
                        i1.teamParticipation = i2.teamParticipation and
                                i1.invitee = i2.invitee and
                                i1.state = Accepted
                ) implies (
                        i2.state ≠ Accepted and i2.state ≠ Pending
                )
        // If a student invited someone to a team, he must be the team creator
        all i: Invite | (some tp: TeamParticipation |
                                tp ≠ i.teamParticipation and
                                        tp in i.teamParticipation.team.teamParticipations
                                            ↪  and
                                        tp.invite = none and
                                        tp.student = i.inviter)
}

enum InviteState { Accepted, Rejected, Pending }

sig TeamParticipation {
        , student: Student
        , invite: lone Invite
        , team: Team
        , studentCommits: Int
} {
        // If the student was invited:
        (invite = none) or (
                        // he must have been invited to this
                        invite.teamParticipation = this
                                // if already subscribed to a battle, he must have
                                    ↪ accepted
                                and (team.participation ≠ none ⟹ invite.state = Accepted
                                    ↪ )
                                // the invitee must be the same as the student
                                and invite.invitee = student)
}

sig Team {
        , participation: lone BattleParticipation
        , teamParticipations: set TeamParticipation
} {
```

```
        // Sync the two relationships
        all tp: TeamParticipation | (tp in teamParticipations) ⇔ (tp.team = this)
        // There must be exactly one student which was not invited
        one p: TeamParticipation | p.team = this and p.invite = none and (
                // All invited students must be invited by that student
                all p1: TeamParticipation | (
                                p ≠ p1 and p1.team = this
                        ) implies (
                                p1.invite ≠ none and p1.invite.inviter = p.student
                        )
        )

        // Can't have the same student twice in the same team
        all disj p1, p2: TeamParticipation | (
                        p1.team = this and p2.team = this
                ) implies p1.student ≠ p2.student
        // If there's the participation, all students must be
        // subscribed to the tournament
        (participation = none) or (
                all st: Student, teamPart: TeamParticipation | (
                                st = teamPart.student and teamPart.team = this
                        ) implies (some subscription: TournamentSubscription |
                                subscription.student = st
                                        and subscription.tournament = participation.
                                            ↪ battle.tournament
                        )
        )
        // If there's a participation, we must respect the battle requirements
        participation = none or (
                        #teamParticipations ≥ participation.battle.minStudents and
                                #teamParticipations ≤ participation.battle.maxStudents
                )
}

sig BattleParticipation {
        , team: Team
        , battle: Battle
        , registrationDate: DateTime
        , score: Int
        , repo: lone Repository
} {
        // Sync the two relationships
        team.participation = this
        // There must be a team for this battle participation
        some team: Team | team.participation = this
        // Must be registered before the deadline
        lte[registrationDate, battle.registrationDeadline]
        // Repo can only exist past the battle REGISTRATION state
        // (if the team created it on their own)
        repo ≠ none ⟹ gt[battle.state, Registration]
        // Sync the two relationships
        repo ≠ none ⇔ repo.battleParticipation = this
}

// A student must only be in one team at a time for each battle
fact studentInOneTeamPerBattle {
        all disj p1, p2: TeamParticipation | (
                        p1.student = p2.student
                                and p1.team.participation ≠ none
                                and p2.team.participation ≠ none
                ) implies (
                        p1.team.participation.battle ≠ p2.team.participation.battle
                )
```

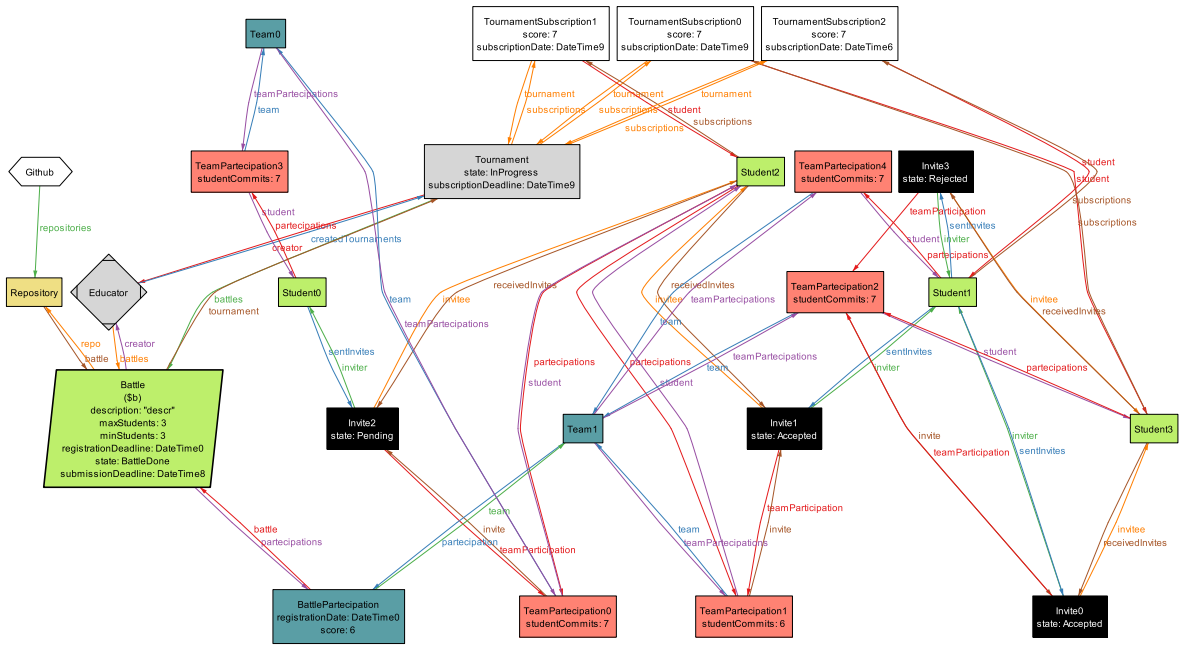
```
}

one sig Github {
        , repositories: set Repository
} {
        all r: Repository | r in repositories
}

sig Repository {
        , battle: lone Battle
        , battleParticipation: lone BattleParticipation
} {
        // Sync the two fields
        battle ≠ none  ⇔  battle.repo = this
        // Sync the two fields
        battleParticipation ≠ none  ⇔  battleParticipation.repo = this
        // Must be used either by a battle or by a team
        battle ≠ none or battleParticipation ≠ none
        not (battle ≠ none and battleParticipation ≠ none)
}
```

## 4.2. Models

### Team of students participate in a battle

This model was generated using:

```
run {
    #Battle = 1
    some b: Battle | b.minStudents = 3 and
                     b.maxStudents = 3 and
                     #b.participations ≥ 1
    #Team = 2
    #Invite = 4
    some i: Invite | i.state = Rejected
    some i: Invite | i.state = Pending
    #Student = 4
    #Educator = 1
    #Badge = 0
} for 10
```




The diagram shows that:

(a) only one student (the team creator) can send invites to join the team (Student0 for Team0 and Student1 for Team1)

(b) if the team has already been subscribed to a battle all the invites must have been accepted

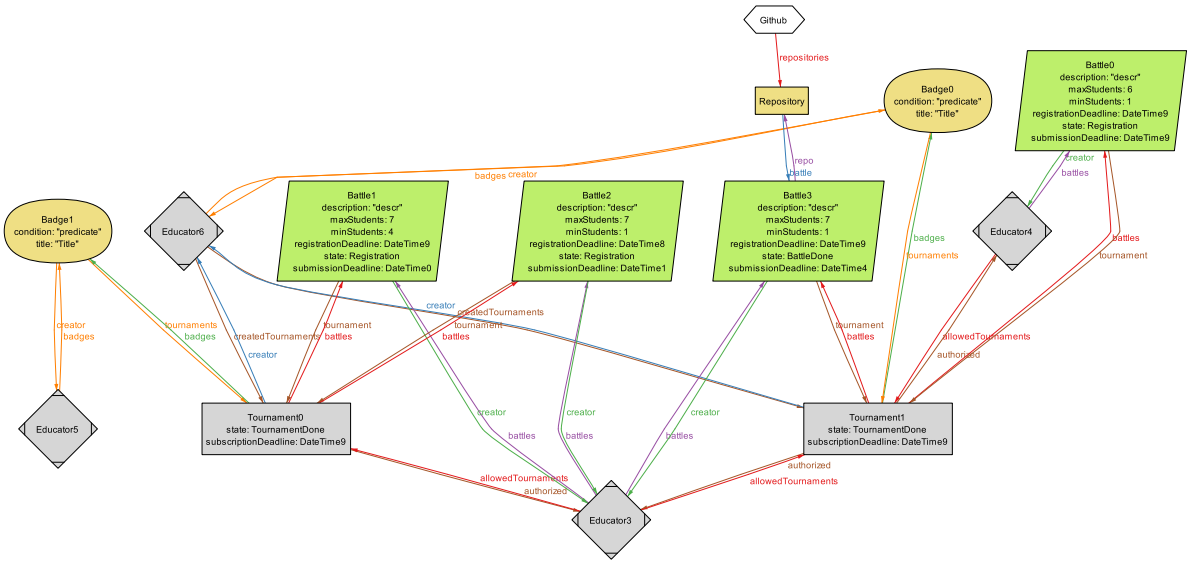
(c) you can receive multiple invites to the same team only if you rejected the previous ones (such as Student3)

(d) you can receive invites for multiple teams, given they are not for the same battle (such as Student2)

### Management of tournaments and battles

This model was generated using:

```
run {
    #Student = 0
    #Tournament = 2
    #Battle ≥ 3
    #Educator ≥ 3
    #Badge ≥ 2
} for 10
```




The diagram shows that:

(a) Educator3 has not created any tournament, yet they are authorized to create battles in both and they have done so

(b) Educator4 has created a tournament and a corresponding battle in it

(c) Educator5 has created a badge unrelated to the tournaments shown here, however it can be reused by other teachers

# 5 | Effort spent

| Member of group | Chapter | Time spent |
|---|---|---|
| **Giacomo Orsenigo** | Introduction | 8h |
| | Overall Description | 5.5h |
| | Specific Requirements | 18h |
| | Formal analysis using alloy | 1.5 |
| **Francesco Ferlin** | Introduction | 6h |
| | Overall Description | 8h |
| | Specific Requirements | 8h |
| | Formal analysis using alloy | 9h |
| **Federico Saccani** | Introduction | 7h |
| | Overall Description | 16h |
| | Specific Requirements | 9h |
| | Formal analysis using alloy | 1.5h |

Table 5.1: Time spent by each member of group.

# Bibliography

[1] S. Chacon and B. Straub. Pro Git - appendix b: Embedding Git in your applications, 2014. URLs: https://git-scm.com/book/en/v2/Appendix-B%3A-Embedding-Git-in-your-Applications-Command-line-Git, https://git-scm.com/book/en/v2/Appendix-B%3A-Embedding-Git-in-your-Applications-Libgit2, https://git-scm.com/book/it/v2/Appendice-B%3A-Embedding-Git-in-your-Applications-JGit and https://git-scm.com/book/en/v2/Appendix-B%3A-Embedding-Git-in-your-Applications-go-git, visited 13/12/2023.

[2] Git. URL https://git-scm.com/.

[3] GitHub. URL https://github.com/.

[4] GitHub GraphQL API. URL https://docs.github.com/en/graphql.

[5] GitHub Rest API. URL https://docs.github.com/en/rest?apiVersion=2022-11-28.

[6] GitHub SSH authentication. URL https://docs.github.com/en/authentication/connecting-to-github-with-ssh.

[7] GraphQL. URL https://www.eclipse.org/jgit/.

[8] JGit. URL https://www.eclipse.org/jgit/.

[9] SAST. A curated list of static analysis (SAST) tools and linters. URL https://github.com/analysis-tools-dev/static-analysis.

[10] SonarQube. URL https://www.sonarsource.com/products/sonarqube/.

[11] SonarQube Web API. URL https://docs.sonarsource.com/sonarqube/latest/extension-guide/web-api/.

[12] Wikipedia contributors. List of tools for static code analysis — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=List_of_tools_for_static_code_analysis&oldid=1189383292, 2023. [Online; accessed 13-December-2023].

# List of Figures

# List of Tables